The Journal of Systems and Software 86 (2013) 2520–2541

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

S-IDE: A tool framework for optimizing deployment architecture of High Level Architecture based simulation systems

Turgay Çelik[a,*], Bedir Tekinerdogan[b]

[a] Department of Computer Engineering, Hacettepe University, Ankara, Turkey
[b] Department of Computer Engineering, Bilkent University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 30 June 2012
Received in revised form 27 February 2013
Accepted 1 March 2013
Available online 20 March 2013

Keywords:
Deployment model optimization
Metamodel based tool development
Distributed simulation
High Level Architecture (HLA)
FEDEP
Software architecture
Model transformations
Metamodeling

ABSTRACT

One of the important problems in High Level Architecture (HLA) based distributed simulation systems is the allocation of the different simulation modules to the available physical resources. Usually, the deployment of the simulation modules to the physical resources can be done in many different ways, and each deployment alternative will have a different impact on the performance. Although different algorithmic solutions have been provided to optimize the allocation with respect to the performance, the problem has not been explicitly tackled from an architecture design perspective. Moreover, for optimizing the deployment of the simulation system, tool support is largely missing. In this paper we propose a method for automatically deriving deployment alternatives for HLA based distributed simulation systems. The method extends the IEEE Recommended Practice for High Level Architecture Federation Development and Execution Process by providing an approach for optimizing the allocation at the design level. The method is realized by the tool framework, *S-IDE* (Simulation-IDE) that we have developed to provide an integrated development environment for deriving a feasible deployment alternative based on the simulation system and the available physical resources at the design phase. The method and the tool support have been validated using a case study for the development of a traffic simulation system.

1. Introduction

Simulation systems are used to simulate real world concepts in different domains such as manufacturing, performance analysis, decision support, virtual exercises and entertainment. There are different reasons for using simulation systems including analysis and testing, cost reduction in development, training, etc. Due to the complexity of the simulated domain very often the simulation is executed across multiple nodes and likewise several different simulators are integrated within a single distributed simulation environment. The reason for distributing the simulation is usually for reducing the execution time of the simulation, enabling geographic distribution of simulation parts, and enabling large simulations with a large number of users (Fujimoto, 1999).

Developing distributed simulation systems is not easy because different simulators might run on different platforms, adopt different data types, use different communication mechanisms, etc. Hence, an important challenge in distributed simulation systems is the integration, reusability and interoperability of the various simulators. To reduce the effort for developing distributed simulations, several standard simulation infrastructures have been introduced including Distributed Interactive Simulation (DIS) (IEEE, 1998), High Level Architecture (HLA) (Kuhl et al., 1999; IEEE, 2010a), and Test and Training Enabling Architecture (TENA) (Noseworthy, 2008). Among these, HLA is an important IEEE and NATO standard specifies a general purpose simulation architecture for realizing interoperable and reusable distributed computer simulation systems (Kuhl et al., 1999; IEEE, 2010a).

One of the important problems in HLA based distributed simulation systems is the allocation of the different simulation modules to the available physical resources. Each deployment alternative represents a different allocation of modules to physical resources and this can be done in many different ways. Further, each deployment alternative will have a different impact on the performance. This problem can be categorized as a *task allocation problem* that has been widely addressed in the literature (Stone, 1977; Lo, 1988; Pirim, 2006; Mehrabi et al., 2009). To solve the task allocation problem different algorithmic solutions have been proposed. Hereby, the algorithms take as input several optimization parameters such as execution cost, communication cost, memory requirement and I/O cost. Based on these input parameters the task allocation algorithms aim to derive feasible allocation of tasks to processors (Stone, 1977; Lo, 1988). The evaluation of the deployment alternative is usually based on expert judgment and postponed to the implementation phase. One cannot always rely on expert judgment because finding experts that have both a broad and specialized

* Corresponding author. Tel.: +90 505 476 8307.
E-mail address: turgaycelik@gmail.com (T. Çelik).

knowledge on the corresponding domains is not easy. Further, human expert judgments can be feasible for small to medium systems but are inadequate for large and complex systems. Moreover, postponing the evaluation of the deployment alternative to the implementation phase, might lead to non-feasible implementations which 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 in the project schedule and increased cost due to the unnecessary rework of the lifecycle artifacts.

The need for early analysis and optimization of the deployment alternatives has also been addressed by the IEEE Recommended Practice for High Level Architecture Federation Development and Execution Process FEDEP (IEEE, 2003). FEDEP describes 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.

To cope with the above problems and address the needs as addressed by FEDEP, we propose a method and our tool framework *S-IDE* (*Simulation-IDE*) that supports the early analysis of deployment alternatives and the automatic generation of the deployment alternatives for HLA based distributed simulation systems. *S-IDE* tool framework consists of several tools based on metamodels that we have developed including Federation Data Exchange Metamodel, Simulation Modules and Publish–Subscribe Relations Metamodel, Physical Resources Metamodel, Simulation Execution Configuration Metamodel, and Deployment Metamodel. Based on the design models developed with these tools, the necessary parameter values for the task allocation algorithms are defined, which are then used for automatic generation of a feasible deployment alternative. In addition, the tool framework can be used for design level analysis including, the impact of adding new simulations modules to the system, suitability of the selected physical resources for the given simulation design, and the impact of the change of publish–subscribe relations. To illustrate the usage of the method and S-IDE we have used a realistic case study concerning the development of a traffic simulation.

The remainder of the paper is organized as follows. In Section 2 we provide the background on HLA and Model Driven Engineering (MDE). Section 3 defines the problem statement based on a case study that will be used in subsequent sections. Section 4 presents the method for evaluating alternative design options briefly. Section 5 describes the metamodels that S-IDE tool framework is built on. Section 6 presents the model transformations step by step for deriving feasible deployment alternatives. Section 7 provides realization of S-IDE tool framework and using S-IDE to derive a feasible deployment alternative for the case study. Section 8 provides the evaluation of the tool. Section 9 provides the discussion. Section 10 describes the related work and finally we conclude the paper in Section 11.

2. Preliminaries

In this section we describe the background for understanding and supporting the approach that we present in this paper. In Section 2.1 we present a brief definition of the High Level Architecture (HLA), followed by a short overview of Model-Driven Engineering (MDE) in Section 2.2.

2.1. High Level Architecture (HLA)

As stated before, HLA is an IEEE standard that supports development of reusable and interoperable distributed simulation systems (Kuhl et al., 1999; IEEE, 2010a,b,c). To support the development of HLA compliant simulation systems, the "Federation Development and Execution Process – FEDEP" has been defined as a part of HLA standard (IEEE, 2003).

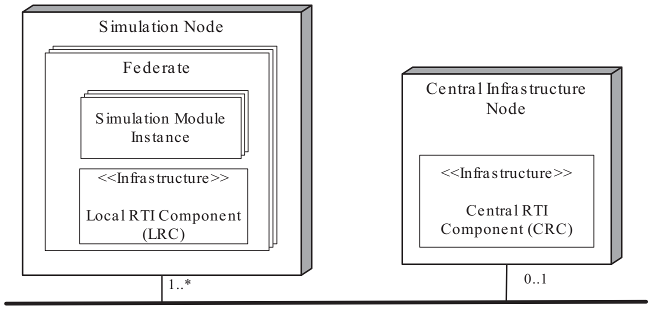

Simulation Node
Federate
Simulation Module Instance
<<Infrastructure>>
Local RTI Component (LRC)
Central Infrastructure Node
<<Infrastructure>>
Central RTI Component (CRC)
1..*
0..1

Fig. 1. Reference architecture for the high level architecture.

Based on a domain analysis to HLA standard we could derive the reference architecture of HLA based simulation systems which is shown in Fig. 1. A typical simulation system is deployed on a number of *Simulation Nodes*. Each *Simulation Node* includes one or more *Federates* which are processes that together form the simulation execution. Each member includes a number of *Simulation Module Instances* and *Local RTI Component* (*LRC*). *Simulation Module Instances* represent objects for simulating entities or events in the simulation. RTI represents the runtime infrastructure that realizes the HLA standard (IEEE, 2010a). *LRC* enables bi-directional interaction between federates for data exchange and collaborative execution of the simulation.

The simulation may also include an optional *Central Infrastructure Node* that contains *Central RTI Component* (CRC) which is responsible for managing the simulation lifecycle, timing, synchronization, and discovery concerns. Although this component is not mandatory, as a convention, major RTI implementations provide *CRC* definitions. In case *CRC* is missing, the services need to be supported by the *LRCs.* As such both the *LRC* and *CRC* provide similar services. In Fig. 1 this is indicated through the stereotype «Infrastructure».

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).

The basic interaction model that is adopted in the HLA 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 Fig. 1. Federates in the simulation execution can publish and subscribe data exchange model elements through the services provided by the «Infrastructure» components. HLA standard defines the *Object Model Template* (*OMT*) that can be used to define different data exchange models which are called *Federate Object Model* (*FOM*) and *Simulation Object Model* (*SOM*) (IEEE, 2010c).

2.2. Model Driven Engineering (MDE)

In traditional, non-model-driven software development the link between the code and higher level design models is not formal but intentional. Required changes are usually addressed manually using the given modeling language. Because of the manual adaptation the maintenance effort is not optimal and as such sooner or later the design models become inconsistent with the code since changes are, in practice, defined at the code level. One of the key motivations for introducing model-driven engineering (MDE) is the

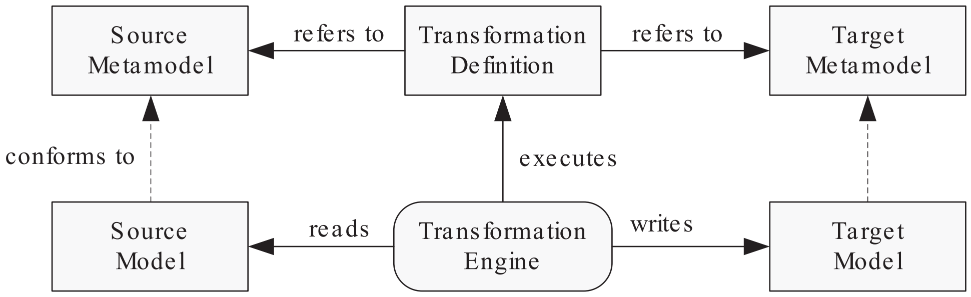


Fig. 2. Model-transformation pattern.

need to reduce the maintenance effort and as such support evolution (Frankel et al., 2004; Bezivin, 2005; Schmidt, 2006). MDE aims to achieve this goal through defining models and metamodels as first class abstractions, and providing automated support using model transformations. For a given change requirement the code is not changed manually but automatically generated or regenerated, thereby substantially reducing maintenance effort. Further, because of the formal links between the models and the code the evolution of artifacts in the model-driven development process is synchronized. The link between the code and models is formal. In fact, there are only models, and as such, 'the documentation is the code'.

MDE requires model transformations to derive the target system from the model (semi)automatically. The general pattern for model transformations is shown in Fig. 2 (Bezivin, 2005; Czarnecki and Helsen, 2006). Here, *Source Model* is provided as input to *Transformation Engine* that generates *Target Model* by using predefined *Transformation Definition*. Both models conform to their respective metamodels.

We can distinguish between Model-to-Model transformations (M2M), Model-to-Text transformations (M2T), and Text-to-Model (T2M) transformations. In M2M the transformation definition refers to metamodels of both the source and the target models. Different M2M approaches have been proposed including, for example, the "Atlas Transformation Language (ATL)" (ATL, 2012) and "Query/View/Transformation (QVT)" (QVT, 2012) tools for model to model transformation (Gronback, 2009). In Model-to-Text Transformation (M2T) the outcome is text such as code or documentation and no target metamodel is used. Examples of M2T approaches are Java Emitter Templates (JET) (JET, 2012) and XPand (XPand, 2012). In Text-to-Model (T2M), the transformation definition refers to metamodels of target models. Examples of approaches that can be used for T2M are XText (XText, 2012) and Grammar to Model Language (Gra2Mol) (Gra2Mol, 2012; Izquierdo and Molina, 2009) that enables model extraction from source code.

In the context of model driven development, Model Driven Architecture (MDA) is an MDE framework defined by the OMG that separates the platform specific concerns from platform independent concerns to improve the reusability, portability and interoperability of software systems (Schmidt, 2006; Frankel et al., 2004). To this end, MDA defines so-called Platform Independent Models (PIMs) and Platform Specific Models (PSMs). The PIM is a model that abstracts from any implementation technology or platform. The PIM is transformed into one or more PSMs which include the platform specific details. Finally the PSM is transformed to code providing the implementation details. Obviously by separating the platform specific concerns and providing mechanisms to compose these concerns afterwards in the code MDA provides a clean separation of concerns and as such the systems are better reusable easier to port to different platforms and have increased interoperability.

3. Problem statement

In this section we define the problem statement and illustrate our approach by using a concrete case study. First subsection defines the case study, second subsection provides a sample scenario build on the case study and finally third section defines the problem by using the defined case study and the scenario.

3.1. Case study—a traffic simulation

The case study that we consider is the development of a traffic simulation. The main objective of this simulation is to support the analysis and optimization of various traffic flow parameters for efficient movement of traffic and minimal traffic congestion problems. The logical view for the case study that depicts simulation environment is given in Fig. 3.

The main participants of the simulation environment are cars, trucks, drivers, speed cameras, traffic lights, lane closes and a traffic analyzer. Other artifacts such as crossings, pedestrians, fixed/mobile radars, on-ramps and weather conditions that affect

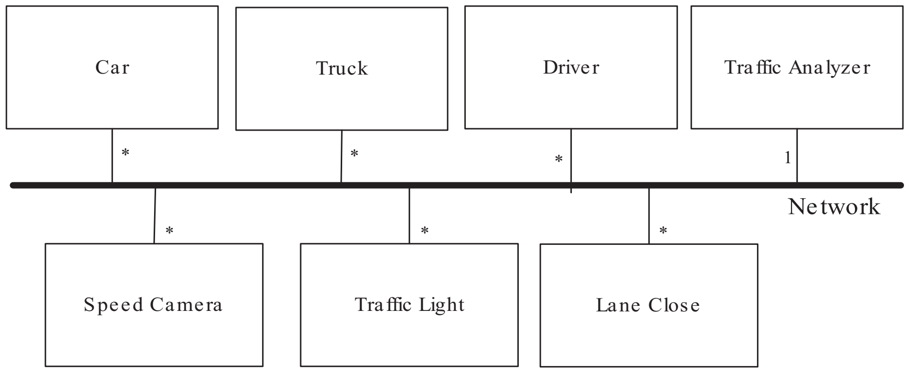


Fig. 3. Logical view of the case study.

Table 1
A sample scenario for the case study.

Simulation module	Number
Car simulation	600
Truck simulation	80
Driver simulation	680
Speed camera simulation	5
Traffic light simulation	15
Lane close simulation	4
Traffic analyzer simulation	1

the traffic flow are not included in the case study for the sake of simplicity. In the figure no particular number for the simulation participants 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 sub-section.

The defined simulation system case study includes cars and trucks as vehicles. A vehicle model shall include properties such as model year, motor power, current driver id, etc. Drivers have different physical and behavioral properties that affect the traffic flow. A driver model shall include properties such as driver id, socio-demographic factors (age, gender, driving experience in years, etc.), driving style (dissociative, anxious, risky, angry, high-velocity, distress-reduction, patient, and careful) (Taubman-Ben-Ari et al., 2004), and accident experience that indicates how many accidents the driver already be involved in Chung and Wong (2010). Speed cameras, traffic lights, and lane closes are participants that generally slow down the traffic flow. A speed camera model shall define position and speed limit value parameters. A traffic light model shall define position and light state (red, yellow, or green). A lane close model shall define a start position, an end position, time slice that lane is closed and a lane index that indicates closed lane (like 1st lane, 2nd lane). Traffic analyzer is a passive participant that collects simulation data from other participants such as vehicles and drivers to perform analysis.

3.2. *A sample scenario for the traffic simulation case study*

After the definition of the simulation environment in the case study section above, we can now define a sample simulation scenario. A scenario includes the types and numbers of major simulation entities according to the earlier defined simulation environment. Table 1 shows a sample scenario for the case study.

The 'Simulation Module' column of the table indicates the simulation participants that together form the simulation of the system. The 'Number' column defines the number of simulation participants of the simulation module type in the given scenario. For example, in the scenario as defined in Table 1 there are 600 cars and 80 trucks. As it can be observed for a given scenario the total number of the required simulation modules might be quite large. For the scenario given in Table 1 total number of simulation modules is 1386.

3.3. *Defining the problem statement*

After the simulation objectives and a sample scenario are defined, we can start designing the simulation system. Using the reference architecture as shown in Fig. 1 and the given scenario in Table 1, we can derive the deployment alternative. A deployment alternative defines the mapping of the simulation modules in the scenario to the nodes and federates.

For example, we can define a deployment alternative with four nodes in which all car simulation modules are deployed on the first node, all truck simulation modules are deployed on the second node, all driver simulation modules are deployed on the third node, and the rest of the simulation modules are deployed on the fourth node. This alternative actually follows the conceptual separation of concerns in which a separate node is logically defined for each simulation module type. Further, the communication overhead among same simulation module types such as cars, trucks, etc. are minimized because of being deployed on same node. Although this alternative is easy to understand because of the logical separation of concerns, it does not always pay-off. This is because separately deployed simulation modules such as car, truck and driver modules may need to interact very frequently with each other for global coordination.

A second example deployment alternative may contain only three nodes. In this alternative car, truck and driver simulation modules are all deployed on first node, the speed camera, traffic light and lane close modules are deployed on second node while traffic analyzer module is deployed on third node separately. This alternative reduces the communication overhead among car, truck and driver simulation modules by deploying all of them on the same node, but on the other hand this deployment configuration may cause resource (memory, processing power, etc.) suffering on this node.

We can derive many more different deployment alternatives which may differ with respect to the number of deployment nodes, the mapping of simulation modules to the federates, etc. Obviously, the number of deployment alternatives is very large and each deployment alternative will perform different with respect to different quality considerations such as logical separation for understandability, optimizing communication overhead, enhancing utilization of physical resources, etc.

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 non-feasible implementations which 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 tool framework for designing the simulation environment and deriving feasible deployment alternatives for HLA based simulation systems.

4. Method for deriving feasible deployment alternatives

In this section we provide the method for deriving and evaluating feasible deployment alternatives briefly before defining the design and the implementation of the S-IDE tool framework. The method will be used in the design phase where the system is not developed yet, and the code is not available.

The process flow of the method is represented as an activity diagram as shown in Fig. 4. Finding a feasible deployment model may require several iterations of process steps. Further, the final deployment model is actually built on several iterations of the design, development, and integration/test activities defined in FEDEP (IEEE, 2003). Hereby, the initial deployment model is prototyped and tested in development and integration/test activities, and the results are fed back to the designer until a satisfactory alternative is derived. The process steps can be briefly explained as follows:

1. *Design Federation Data Exchange Model.* This step defines an initial version of the Federation Data Exchange Model (FDEM) that is necessary to enable data exchange among simulation modules. Actually, a FDEM is an extended version of an HLA

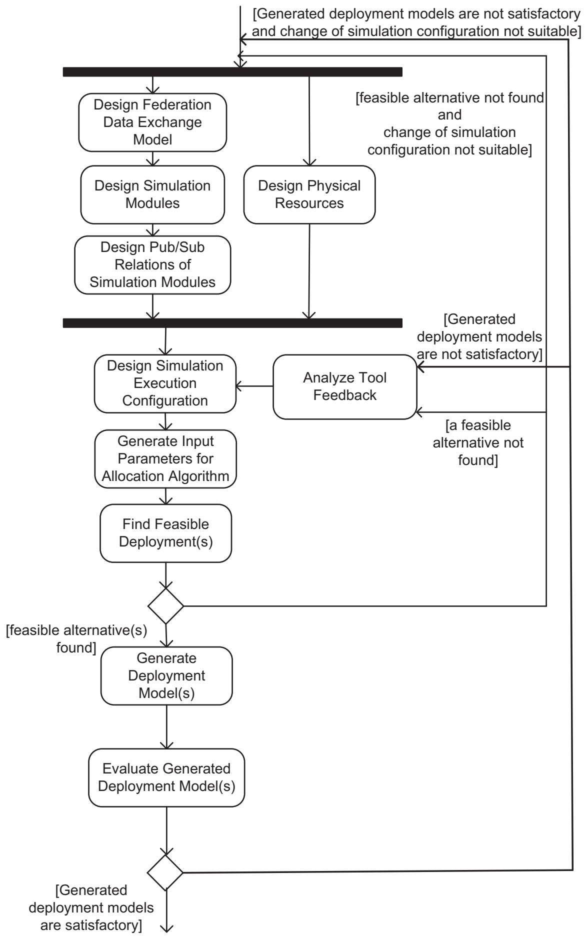

[Generated deployment models are not satisfactory and change of simulation configuration not suitable]
Design Federation Data Exchange Model
[feasible alternative not found and change of simulation configuration not suitable]
Design Simulation Modules
Design Physical Resources
Design Pub/Sub Relations of Simulation Modules
[Generated deployment models are not satisfactory]
Design Simulation Execution Configuration
Analyze Tool Feedback
Generate Input Parameters for Allocation Algorithm
[a feasible alternative not found]
Find Feasible Deployment(s)
[feasible alternative(s) found]
Generate Deployment Model(s)
Evaluate Generated Deployment Model(s)
[Generated deployment models are satisfactory]

Fig. 4. Method for deriving feasible deployment alternatives.

Federation Object Model (FOM) (IEEE, 2010c). Details of this extension relation will be explained in Section 5.

2. *Design Simulation Modules*. This step includes the definition of simulation modules that are artifacts of a simulation system responsible for modeling each part of the system. In the given example scenario as given in Table 1 simulation modules are, for example, Car, Truck, Driver, Speed Camera, etc.
3. *Design Pub/Sub Relations of Simulation Modules*. This step defines the publish/subscribe relations of simulation modules based on the Federation Data Exchange Model which is defined in first step of the process. For example, a *Car* object can be published by *Car* Module and subscribed by *SpeedCamera* Module.
4. *Design Physical Resources*. Parallel to the above three steps, this step 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 4 nodes on which the simulation participants need to be deployed. Further it could be decided that each node has a memory capacity of 36,840 MB and contains two processing units at the frequency of 3.0 MHz. Equally, the nodes could also have different memory capacity and processing power.
5. *Design Simulation Execution Configuration*. This step defines the run-time 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.
6. *Generate Input Parameters for Allocation Algorithm*. After the steps above, both the static and run-time properties of the simulation participants, the simulation entities and the physical resources are defined, this step derives the necessary parameter values for the algorithms that define a feasible deployment alternative.
7. *Find Feasible Deployment Model(s)*. This activity takes the outputs of the previous activity as input parameters and executes the algorithms to compute feasible deployment alternatives. If a feasible deployment is found, this activity yields a table that represents the mapping of tasks (module instances) to processors (nodes). It is also possible to generate more than one feasible deployment alternative and present the results to the designer for deciding the deployment model.
8. *Analyze Tool Feedback*. If no feasible solution was found in the previous step, detailed feedback is presented to the designer to optimize the design model. The designer will first try to update the simulation execution configuration. If a feasible deployment can still not be found then the designer can decide to return to the beginning of the process to refine/update the design.
9. *Generate Deployment Model(s)*. The task-processor mapping tables that are the output of the previous step will be used in this step generate one or more deployment models.
10. *Evaluate Generated Deployment Model(s)*. In this step, the designer evaluates the generated deployment model by comparing it with: (1) other deployment models generated by the selected CTAP algorithm (2) generated alternatives with other CTAP algorithms (3) manually generated deployment models with expert judgment. The *S-IDE* tool provides automatic analysis and comparison features that enable evaluating deployment models with respect to different quality factors. The generated deployment models will be improved until they are considered to be satisfactory with respect to the defined goals of the designer. Here a satisfying alternative defines an alternative that meets the expected improvement rate of the communication and execution costs for the deployment model. If a satisfactory solution is found, the feasible deployment alternative derivation process will end. Otherwise, the designer can generate design diagnostic feedback report with the *S-IDE* tool and analyses the provided feedback. The designer first tries to find a satisfying deployment alternative by updating the simulation execution configuration. If updating the simulation execution configuration is not enough to achieve a satisfying deployment alternative, the designer can decide to return to the beginning of the process to refine/update the design.

5. Metamodels

In this section we will describe the metamodel for the models that are defined in the method as shown in Fig. 4. The metamodel is shown in Fig. 5. As it can be seen from the figure the metamodel consists of five main parts including *Federation Data Exchange, Simulation Modules and Publish/Subscribe Definitions, Physical Resources, Simulation Execution Configuration* and *Deployment* metamodels. We explain each of these metamodels in the following subsection.

5.1. Federation data exchange metamodel

The *Federation Data Exchange Metamodel* is used to describe *Federation Data Exchange Models* in Step 1 of the method described in Section 4. We have defined this metamodel by reusing and extending the HLA OMT (IEEE, 2010c) standard which defines a standard metamodel for deriving Federation Object Models (FOM) and Simulation Object Models (SOM). The resulting Federation Data

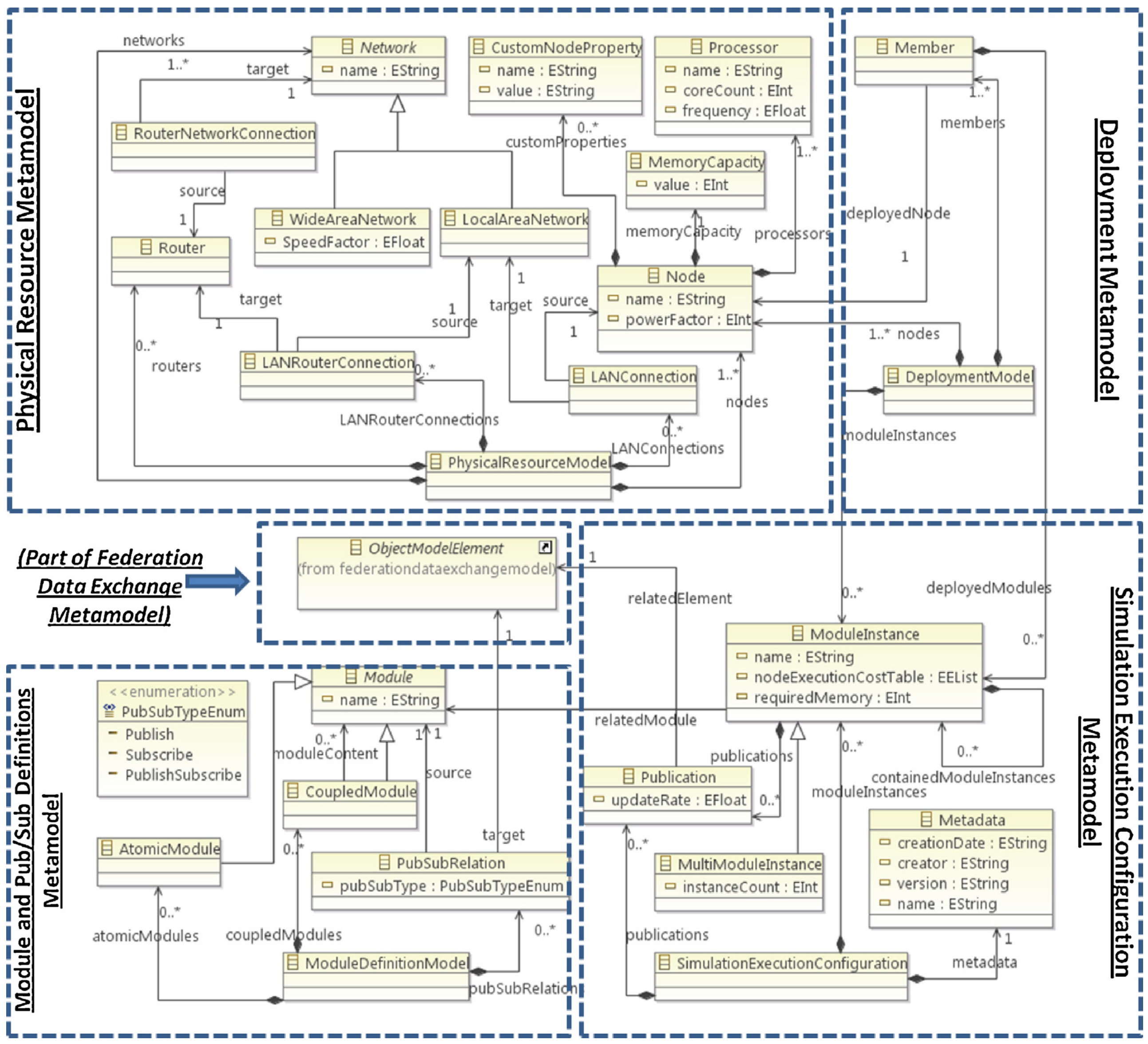


Fig. 5. High level view of the metamodels.

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. Since the resulted metamodel is quite large in size, we have only shown the part of the metamodel that relates to the other parts of our metamodel in Fig. 5. As it is shown in the figure the element *ObjectModelElement* is the part that defines the connection with the other parts of our metamodel.

To represent simulation entities, HLA OMT specification defines the three key elements of *ObjectClasses*, *Interactions* and *DataTypes* (not shown in the figure). *ObjectClasses* are used to define the simulation entities. In our case, *ObjectClasses* are used to represent, for example, *Car*, *Truck*, *Speed Camera*, etc. *Interactions* are used to represent the messaging semantics among simulation participants. For example, messages like *SpeedLimitViolation*, *TrafficLightChange* are examples of interactions. Finally, *DataTypes* represent types of the attributes of *ObjectClasses* and parameters of *Interactions*. For example, the *ObjectClass Car* could have an attribute *position* of type *Position2D*, and the Interaction *SpeedLimitViolation* can have a parameter *carID* of String type.

5.2. Modules and publish/subscribe relations metamodel

The *Simulation Modules and Publish/Subscribe Relations Metamodel* is used to describe Simulation Modules and Simulation Publish/Subscribe Models in steps 2 and 3 of the method described in Section 4. We have defined a common metamodel that can be used to define both the simulation modules and the composition relations. Similar to the Discrete Event Virtual Simulation (DEVS) specification (Zeigler, 2003) the metamodel defines atomic and coupled models that form the simulation systems.

As shown in Fig. 5, *ModuleDefinitionModel* represents a module definition model that defines modules and their Publish/Subscribe relations. *ModuleDefinitionModel* contains elements of *AtomicModule*, *CoupledModule* and *PubSubRelation*. 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. *Module* is the abstract base class for *AtomicModule and CoupledModule* definitions. *PubSubRelation* class in the metamodel defines a publish/subscribe relation between a simulation module *Module* and

Federation Data Exchange Model (FDEM) element *ObjectModelElement.*

5.3. Physical resources metamodel

The *Physical Resource Metamodel* is used to represent the artifacts for modeling the available physical resources in Step 4 of the method described in Section 4.

PhysicalResourceModel defines a physical resource model which can have one or more *Nodes* that represent computation resources. The *powerFactor* attribute defines the processing power of the node relative to other nodes. A node can have one or more processors, memory capacity, and one or more custom node properties. *Processor* defines properties of a processing unit using the attributes *name, frequency* and *coreCount*. *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 as name-value pairs (e.g. *diskCapacity* – 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. *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. *LANRouterConnection* class represents connection of a LAN to a router while the *RouterNetworkConnection* class represents connection of a router to a network.

5.4. Simulation Execution Configuration Metamodel

The *Simulation Execution Configuration Metamodel* is used to define the artifacts to model the simulation execution configuration in Step 5 of the method described in Section 4. *SimulationExecutionConfiguration* class defines a simulation execution configuration which contains elements of *Metadata, ModuleInstance, MultiModuleInstance*, and *Publication*. *Metadata* defines *name*, *version*, *creator*, and *creation date* of a simulation execution configuration. *ModuleInstance* represents an instance of a simulation module that is defined in the *Simulation Modules and Publish*/Subscribe *Relations Metamodel.*

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 *SpeedCamera* model changes according to existing *Cars* and *Trucks* in the execution configuration. The execution cost is a scaled value that shows the execution cost of a Simulation Module Instance in comparison with other Simulation Module Instances in the execution configuration. For example, the execution cost for each *Car* module instance is defined using scaled value and defined as 7 over 20 for one node, 14 over 20 for another node, etc. The execution costs of simulation modules are influenced by the processor's *powerFactor* and *memoryCapacity* attributes. In a similar sense, the communication costs among simulation modules are influenced by the networks *speedFactor* attribute. Since the execution and communication costs of module instances can only be exactly measured after the system is developed (Lauterbach et al., 2008), during design time their values can only be estimated. This estimation can be conducted by using, for example, design phase complexity calculation methods such as proposed by Podgorelec and Hericko (2007), or prototyping.

The attribute *requiredMemory* of *ModuleInstance* represents the estimated memory amount that the module instance will require during execution. Similar to the execution cost, this parameter can be estimated in the design phase. The attribute *instanceCount* of *MultiModuleInstance* 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 traffic scenario, there can be hundreds of *Cars* 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 module instances that a coupled module contains. 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 FDEM. Each publication is associated with an object class attribute set or an interaction class defined in FDEM.

The *updateRate* attribute shows how many times a module instance will update a FDEM element in a second. For example, we could decide to have 1000 *Car* module instances where each of them publishes a *Car* object with update rate of 2 times per second.

5.5. Deployment Metamodel

The *Deployment Metamodel* is used to describe the deployment model in Step 8 of the method described in Section 4. The deployment Metamodel 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. Model transformations

The method in Section 4 has been realized as a set of model transformations. The model transformation chain is shown in Fig. 6. This model transformation chain consists of the three basic transformations *Models-to-CTAP-Params Transformations*, *CTAP Solver*, and *TaskAlloc-to-Deployment Model Transformation*. These transformations are generic and do not depend on the use of a particular CTAP algorithm. We explain these transformations in the following subsections.

6.1. Manual design of simulation models

The process starts with defining the federation data exchange model, simulation modules and pub-sub relations models, physical resources model, and simulation execution configuration model. These are the outputs of the first five activities of the method defined in Section 4. Each of the models conforms to their corresponding metamodel, which were described in Section 5.

6.2. Models-to-CTAP parameters transformation

The simulation models are provided to the model transformation *Models-to-CTAP_Params* which generates inputs for the "Capacitated Task Allocation Problem (CTAP)" (Pirim, 2006; Mehrabi et al., 2009) algorithm. The CTAP is a refinement of the task allocation problem (TAP) to which it adds constraints such as memory capacity and processing power to the problem formulation. The objective in the CTAP is to minimize the sum of total execution cost and total communication cost among the simulation module instances. Hereby, the memory capacity and the processing power of each node should not be exceeded.

The metamodel for CTAP parameter specification is given in Fig. 7. In fact, the required parameters of CTAP can be extracted from the simulation design that has been defined in the previous activities. In Table 2 we describe each parameter and how it is extracted from the design. These parameters are independent of the various CTAP algorithm implementations. The transformation

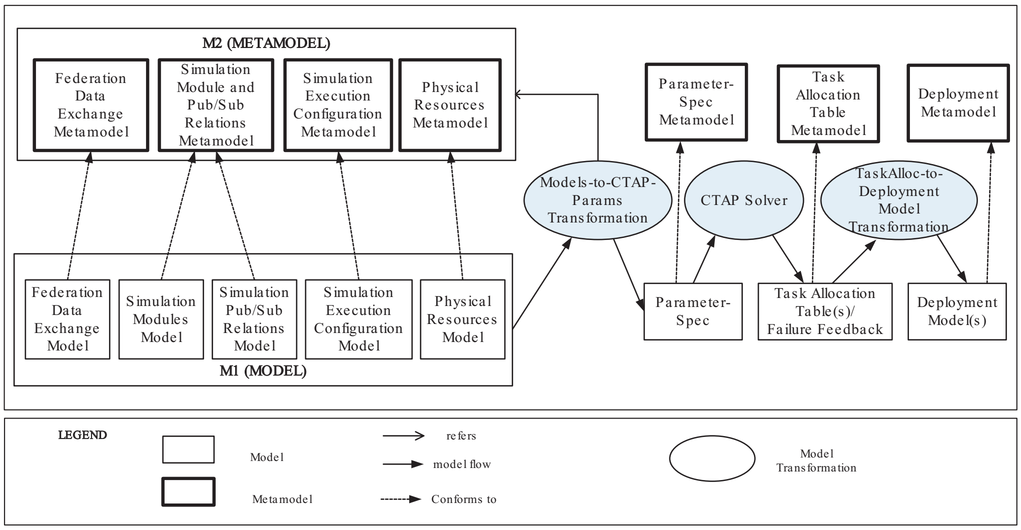

M2 (METAMODEL)
Federation Data Exchange Metamodel
Simulation Module and Pub/Sub Relations Metamodel
Simulation Execution Configuration Metamodel
Physical Resources Metamodel
Parameter-Spec Metamodel
Task Allocation Table Metamodel
Deployment Metamodel
Models-to-CTAP-Params Transformation
CTAP Solver
TaskAlloc-to-Deployment Model Transformation
Federation Data Exchange Model
Simulation Modules Model
Simulation Pub/Sub Relations Model
Simulation Execution Configuration Model
Physical Resources Model
M1 (MODEL)
Parameter-Spec
Task Allocation Table(s)/ Failure Feedback
Deployment Model(s)
LEGEND
Model
Metamodel
refers
model flow
Conforms to
Model Transformation

Fig. 6. Model-transformation chain that realizes the method for deriving feasible deployment alternative.

Models-to-CTAP-Params Transformations performs a generic transformation to extract these CTAP parameter values that can be used by the selected CTAP algorithm implementations.

6.3. *CTAP solver*

The input parameters that were generated in the previous step are provided to the *CTAP Solver* which aims to find a task-processor allocation. The CTAP Solver does not mandate the use of a particular CTAP algorithm implementation. We have provided a generic mechanism to enable the selection and adaptation of different CTAP algorithm implementations in the *CTAP Solver*. For this we have used the OSGI service registry capabilities of the Eclipse Equinox platform (McAffer et al., 2010; OSGI, 2011; Equinox, 2012). The S-IDE tool defines a generic service interface plug-in that can be realized by various plug-ins to provide specific CTAP algorithm implementations. The *CTAP Solver* module queries the registered CTAP algorithm implementations via OSGI service registry and as such enables the user to select one of the registered alternative algorithms. For detailed information on adding new CTAP algorithm implementations we refer to the project web site (SIDE, 2012).

For our problem, we focus on optimizing the allocation of simulation module instances to nodes by considering execution cost, memory requirement, communication cost, processing power, and memory capacity parameters as defined in the simulation design. Please note that we do not focus on a particular algorithm but recommend using a practical one for the corresponding case. The output of the *CTAP Solver* is *Task Processor Allocation Table* which describes the mapping of tasks to processors. The metamodel for *Task Processor Allocation* is given in Fig. 8.

For different simulation contexts the designer can choose different CTAP Solver implementations. To support the designer in selecting the appropriate algorithm implementations, the S-IDE

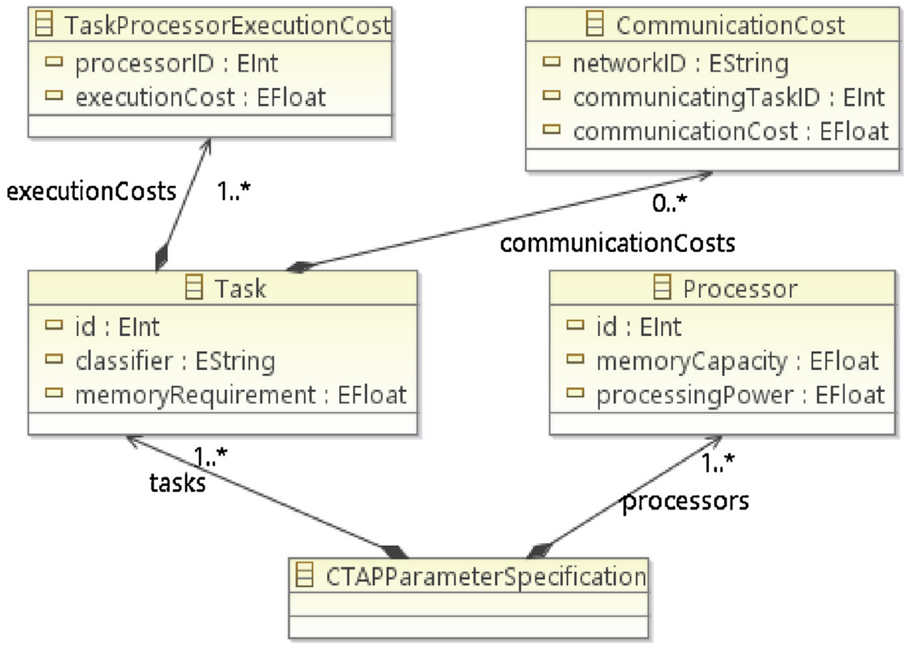


Fig. 7. CTAP parameter specification metamodel.

Table 2
Extracting CTAP parameters from the design.

CTAP parameter	Extraction from design
T	Set of m tasks. Tasks are extracted from module instances defined in Simulation Execution Configuration Model.
P	Set of n non-identical processors. Processors are extracted from *nodes* defined in Physical Resource Model.
M_p	Memory capacity of processor p. Memory capacity is extracted from *memoryCapacity* attribute of each *node* defined in Physical Resource Model.
C_p	Processing power of processor p. Processing power is extracted from *powerFactor* attribute of each *node* defined in Physical Resource Model.
m_i	Amount of memory needed for task i. Amount of required memory is extracted from *requiredMemory* attribute of *ModuleInstance* defined in Simulation Execution Configuration Model.
x_{ip}	Processing power cost of executing task i on processor p. Processing power cost is extracted from *nodeExecutionCostTable* attribute of *ModuleInstance* defined in Simulation Execution Configuration Model.
c_{ij}	Communication cost c_{ij} if tasks i and j are assigned to different processors calculated by using: *Publications* defined in Simulation Execution Configuration Model, *Subscriptions* defined in Publish/Subscribe Relations Model, *Object model elements* defined in Federation Data Exchange Model Communication cost between two nodes is negligible if two tasks are assigned to the same processor.

tool provides the objectives and characteristics of the algorithm that are defined by the algorithm developer.

The CTAP parameters can be prioritized if the selected CTAP algorithm supports such a prioritization. For example, the genetic algorithm based CTAP implementation that we have used for our experiments defines the same coefficients for execution and communication costs, thus the parameter priorities are equal. However, as stated before the S-IDE tool does not mandate a particular implementation of the algorithm, and if needed different implementations might be selected that support the prioritization. The S-IDE tool asks the designer to define the priorities if the selected CTAP algorithm supports the prioritization of the parameters.

6.4. Task allocation-to-deployment model transformation

Task Processor Allocation Table generated in previous step is provided as an input to *Task Allocation-to-Deployment Model Transformation* that generates the final deployment models. In case the *CTAP Solver* cannot find a feasible task to processor allocation alternative or if the alternative is not satisfying, the designer can use the S-IDE Design Analysis Tool that provides a detailed design diagnostic feedback. The design diagnostic feedback contains the following information:

- The communication costs among simulation module instances ordered by size of transferred data per second.
- The simulation data exchange model objects ordered by size.
- The simulation module instances ordered by required amount of memory.
- The physical resources ordered by capacity limits.

The designer can use this diagnostic feedback to analyze the simulation design, update the models, and restart the transformation process again until a feasible and satisfying solution is found. In general, distributed systems are optimized using design heuristics for reducing either the cost parameter values such as bandwidth usage and/or enhancing the capacities of the adopted physical resources (Izosimov et al., 2005; White and Schmidt, 2010). Based on these general design optimization heuristics as well as our own lessons learned from real industrial HLA based distributed simulation systems (Çelik et al., 2012) and OMG DDS based real time systems we have defined the following categories of heuristic rules that can be applied in the method to optimize the system if a feasible task to processor allocation cannot be found:

1. *Simulation Execution Configuration Optimizations*.The designer may first try to reduce update rates in the simulation execution configuration. For example, for the given traffic case study, the designer may decide that trucks are slow vehicles and their update rates in the simulation can be reduced from 2 updates/second to 1 updates/second.
2. *Simulation Design Optimizations*:
(a) If reducing the update rates in the simulation execution configuration does not help finding a feasible solution, the designer can re-organize the subscribed data sets and split the federation data exchange model object structures. In many cases the subscriber only requires a specific set of data class attributes (e.g. speed of the object).
(b) The designer may check the reliability levels of the data exchange model elements. In HLA, a FOM object can be shared among federates either with *Reliable* or *Best Effort* reliability levels. The communication cost of sharing reliable data is higher than best effort sharing. The reliability level of the FOM objects defined *Reliable* could be reduced to *Best Effort* where possible to further optimize the simulation design. For example in the traffic simulation case study, the position of the vehicles is frequently updated and can be defined as *Best Effort*. In such a case, the subscribers shall use dead reckoning methods (Fujimoto, 1999) for calculating the vehicle positions.
3. *Physical Resources Model Enhancements*:If all design level optimizations described above are applied and it is still not possible to find a feasible deployment alternative, the only alternative is enhancing the physical resources.

The design diagnostic feedback is automatically generated if at least one deployment alternative cannot be found. The designer can also manually trigger design diagnostic process in *S-IDE* tool if he/she is not satisfied with the quality of the generated deployment models. In this case, the designer can follow the heuristics listed above to improve simulation design until a satisfying deployment alternative can be derived.

6.5. Implementation and verification of the transformation rules

In the above transformations we have basically applied model-to-model transformations in which the transformation

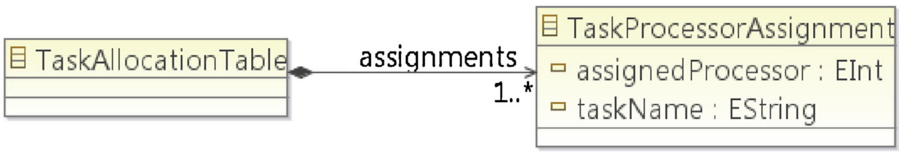


Fig. 8. CTAP task processor allocation metamodel.

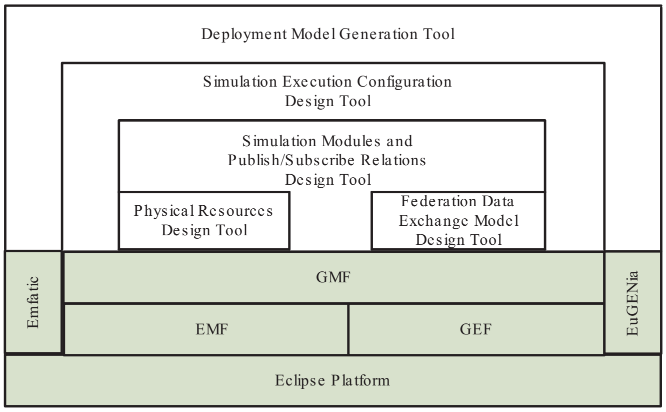

Deployment Model Generation Tool
Simulation Execution Configuration Design Tool
Simulation Modules and Publish/Subscribe Relations Design Tool
Physical Resources Design Tool
Federation Data Exchange Model Design Tool
Emfatic
GMF
EuGENia
EMF
GEF
Eclipse Platform

Fig. 9. Layered architecture of S-IDE environment.

rules refer to metamodels of the source and target models. There are different approaches for implementing model-to-model transformations including direct manipulation, structure-driven, operational, template-based, relational and graph transformation-based and hybrid approaches (Czarnecki and Helsen, 2006). We preferred to adopt the direct model manipulation with Eclipse EMF in which an internal model representation plus some API is provided to manipulate the models. The advantage of direct manipulation approach is that we could implement complex transformation rules using our adopted programming language (Java). Likewise we could more flexibly implement the complex functionality of the model transformations such as calculating communication cost parameters.

Validating the implemented transformation rules can also be done in different ways based on the selected model transformation approach (Büttner et al., 2011). Since we used direct model manipulation approach we could verify the correctness of the rules using unit testing capabilities of the Java programming environment (JUnit, 2012). We have tested each model-transformation unit using different inputs.

7. S-IDE tool framework

In this section we will present the *S-IDE* tool framework which provides an integrated development environment for supporting the method as defined in Section 4 (SIDE, 2012). *S-IDE* tool framework is based on the metamodels as defined in Section 5 and the model transformations as defined in Section 6. *S-IDE* is built on the Eclipse platform and is implemented as a set of plug-ins. The developed plug-ins are built on other Eclipse framework plug-ins including Eclipse Modeling Framework (EMF) (Budinsky et al., 2003), Graphical Editing Framework (GEF) (Moore et al., 2004), and Graphical Modeling Framework (GMF) (Voelter et al., 2006). EMF is a modeling framework and code generation facility that we use to develop the metamodels. GEF is a framework that is used for generating rich graphical editors and views. GMF is a generative component and runtime infrastructure that we use for developing graphical editors for the developed metamodels. Further, we use Emfatic (Daly, 2004), which provides a text editor and a language for editing EMF models. In addition we use EuGENia (Kolovos et al., 2010) GMF tool that provides mechanisms for abstracting away the complexity of GMF and for easier development of GMF editors. EuGENia tool is a part of Epsilon project (Kolovos et al., 2006). The layered tool architecture of the S-IDE is given in Fig. 9.

In the following subsections we describe the top-level tool architecture (Section 7.1), show the application of S-IDE for designing the simulation models for the selected case study (Section 7.2), and describe the generation of the deployment model for the case study (Section 7.3).

7.1. Tool architecture

S-IDE consists of five different tools. The common perspective of *S-IDE* tools is given in Fig. 10. The left pane includes the Model Navigator that shows the available models and their elements. The Model Editing Pane in the middle provides the main drawing area for the simulation design. The Item Palette on the right provides the objects and the connections that are used for creating a design model. The items in this palette can be added to the Editing pane by dragging and dropping. The Properties View at the bottom provides an editing area for the attributes of the design model elements that are selected from the Editing Pane or the Model Navigator.

7.2. Using S-IDE to design simulation models for the case study and derive a feasible deployment

In this section we use the *S-IDE* to design the traffic case study defined in Section 3.2 and we derive a feasible deployment model for the case study. Rest of this section explains each step of using *S-IDE* for the case study.

7.2.1. Designing traffic simulation federation data exchange model

Figs. 11 and 12 together show *Traffic Simulation Federation Data Exchange Model (FDEM)* that has been designed using the *S-IDE* framework. Fig. 11 defines the object classes of data model while Fig. 12 defines the interaction classes. Both figures also define the necessary data types.

In Fig. 11, the *HLAObjectRoot* object class is defined as root class for all other object classes in conformance with HLA OMT standard. *PhysicalEntity* object class derives from the root class and defines two basic properties – *position* and *identification* – of a physical entity. *Position* attribute of *PhysicalEntity* is defined by *Position2D* data type which provides location information in means of *latitude* and *longitude* values. *Car*, *Truck*, *TrafficLight*, *SpeedCamera* and *Driver* object classes are defined in a similar fashion with necessary data type definitions such as *DrivingStyleEnum* enumerated value that represents driving characteristics or *TraficLightEnum* that specifies current light state, one of *Red*, *Yellow*, and *Green* values.

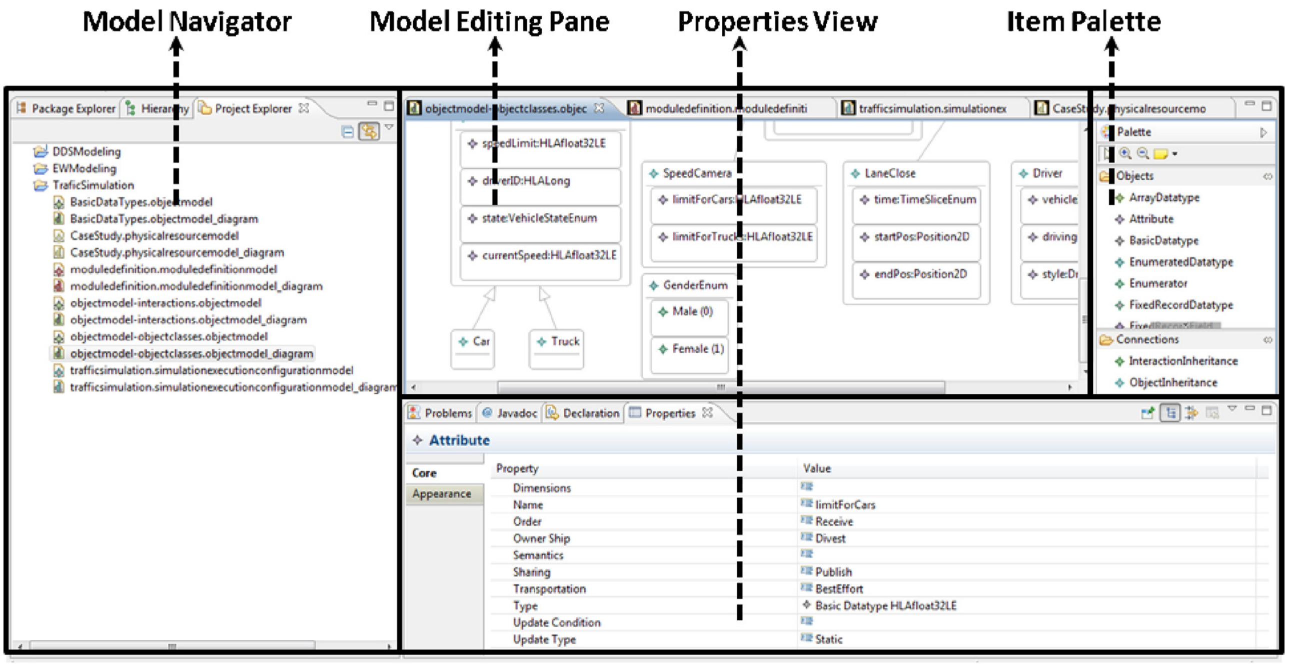
In Fig. 12, the *HLAInteractionRoot* interaction class is defined as root class for all other interaction classes in conformance with HLA OMT standard. Speed limit violations, traffic light violations and accidents are defined as interactions. Fig. 12 also defines various parameters such as violating vehicle id or vehicles/pedestrians that are involved in the accident.

7.2.2. Designing traffic simulation modules and publish/subscribe relations

Fig. 13 shows the design of traffic simulation modules and publish/subscribe relations in means of Traffic Simulation Federation Data Exchange Model defined in previous step.

As shown in figure, *CarModel*, *TruckModel*, *DriverModel*, *TrafficLightModel*, *LaneCloseModel*, and *SpeedCameraModel* simulation modules defined in according to case study. *TrafficAnalyzer* simulation module defined in the case study is refined and *DriverTracker*, *VehicleTracker*, *AccidentTracker*, and *RuleViolationTracker* sub-modules are defined as artifacts of *TrafficAnalyzer* module. This decomposition makes the *TrafficAnalyzer* module a “coupled module” that is composed of several “atomic modules” (see “Modules and Publish/Subscribe Relations” metamodel given in Section 5.2 for atomic and coupled module definitions).

Each module publishes the object and interaction classes that they own modeling responsibility (e.g. *CarModel* publishes *Car*


Model Navigator
Model Editing Pane
Properties View
Item Palette
Package Explorer
Hierarchy
Project Explorer
DDSModeling
EWModeling
TraficSimulation
BasicDataTypes.objectmodel
BasicDataTypes.objectmodel_diagram
CaseStudy.physicalresourcemodel
CaseStudy.physicalresourcemodel_diagram
moduledefinition.moduledefinitionmodel
moduledefinition.moduledefinitionmodel_diagram
objectmodel-interactions.objectmodel
objectmodel-interactions.objectmodel_diagram
objectmodel-objectclasses.objectmodel
objectmodel-objectclasses.objectmodel_diagram
trafficsimulation.simulationexecutionconfigurationmodel
trafficsimulation.simulationexecutionconfigurationmodel_diagram
objectmodel-objectclasses.objec
moduledefinition.moduledefiniti
trafficsimulation.simulationex
CaseStudy.physicalresourcemo
speedLimit:HLAfloat32LE
driverID:HLALong
state:VehicleStateEnum
currentSpeed:HLAfloat32LE
Car
Truck
SpeedCamera
limitForCars:HLAfloat32LE
limitForTrucks:HLAfloat32LE
GenderEnum
Male (0)
Female (1)
LaneClose
time:TimeSliceEnum
startPos:Position2D
endPos:Position2D
Driver
vehicle
driving
style:Dr
Palette
Objects
ArrayDatatype
Attribute
BasicDatatype
EnumeratedDatatype
Enumerator
FixedRecordDatatype
Connections
InteractionInheritance
ObjectInheritance
Problems
Javadoc
Declaration
Properties
Core
Appearance

Attribute

Property	Value
Dimensions	
Name	limitForCars
Order	Receive
Owner Ship	Divest
Semantics	
Sharing	Publish
Transportation	BestEffort
Type	Basic Datatype HLAfloat32LE
Update Condition	
Update Type	Static

Fig. 10. General perspective of S-IDE tool.

object class and *AccidentInteraction* interaction class). Modules also subscribe to object and interaction classes to receive necessary updates of interested data. For example, *DriverModel* subscribes to *TrafficLight* and *Vehicle* object classes for modeling behavior of the driver.

7.2.3. *Designing physical resource model*

Fig. 14 shows an example Physical Resource Model for the case, which has been designed using the Physical Resource Design Tool. In the example, we have defined 4 nodes with different processors and memory capacities. As shown in the figure some nodes, like Node-4, can have more than one processor. Although, the example shows only one Local Area Network on which the nodes are connected, the tool also enables the design of heterogeneous LAN/WAN networks.

7.2.4. *Designing traffic simulation execution configuration*

The Module and Publish–Subscribe Relations Model (Fig. 13) and Physical Resource Model (Fig. 14) are used in the Simulation Execution Configuration Design Tool to define the Simulation Execution Configuration. Part of the latter model is shown in Fig. 15. Here we show an example simulation execution configuration for the scenario as defined in Table 1. The simulation module instances are shown using rectangles. The number of instances for the corresponding module is shown between brackets. For example, in the figure it is indicated that SpeedCameraModel has 5 instances in accordance to the earlier scenario. Note, however, that in this model the scenario is further refined. More specifically, in Table 1 it is indicated that should be a Traffic Analyzer module. In the Simulation Execution Configuration in Fig. 15, Traffic Analyzer module instance contains four sub module instances (AccidentTrackerModel, VehicleTrackerModel, etc.) just like it is defined in Module and Publish–Subscribe Relations Model (Fig. 13). The instances also show the publication properties (published FDEM element and update rate) as shown in figure. For example CarModelInstance publishes Car object class 2 times/second.

7.3. *Generating the deployment models for the case study*

So far, the input models for generating feasible deployment alternatives have been developed manually. Based on these models, feasible deployment alternatives are automatically generated. The top-level algorithm that is used for the automatic generation is shown in Fig. 16.

As stated in line 1, the algorithm *GENERATE_FEASIBLE_DEPLOYMENTS* takes two input parameters: a *physical resource model* and *a simulation execution configuration* as defined, for example, in Figs. 14 and 15, respectively. Line 2 extracts processors from the physical resource model by calling *EXTRACT_PROCESSORS* in which a processor is created for each node in the physical resource model. In Line 3, tasks are extracted from the simulation execution configuration by calling *EXTRACT_TASKS* in which a task is created for each module instance and execution cost among tasks is calculated. In Line 4, the actual CTAP algorithm is executed by calling *EXECUTE_CTAP*. The result of this is stored in *assignment_tables* that includes a list of assignments of tasks to the processors. Likewise, each member of *assignment_tables* defines an abstract specification of a feasible deployment alternative. In Line 5, the deployments are actually generated by calling *CREATE_DEPLOYMENT_MODELS* with the parameter *assignment_tables*.

As shown in the pseudo code of Fig. 16 the CTAP algorithm can generate multiple deployment alternatives. Two samples of deployment alternatives that are generated by the CTAP algorithm are shown in Figs. 17 and 18. The figures represent feasible deployment models for the case study as described in Table 1. As it can be observed from the figures each deployment model includes 4 nodes as it was given before in the physical resource definition model in Fig. 14. Further, the execution configuration model as defined in Fig. 15 has been deployed to the physical nodes to optimize the values for the metrics execution cost, communication cost and memory requirements (see Section 6.2). As it can be also observed from the figures, the two deployment alternatives include different number and types of deployed simulation module instances per node.

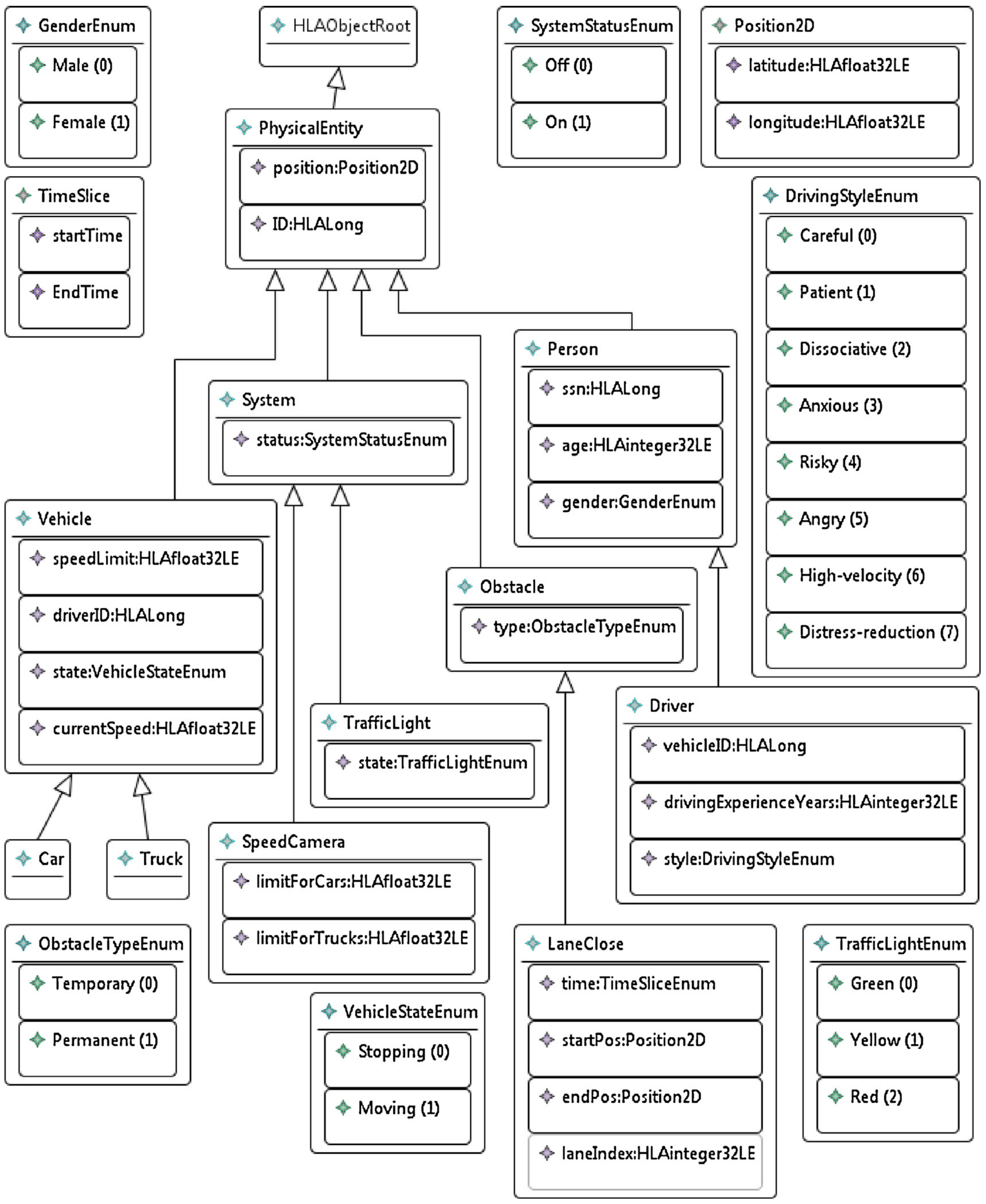


Fig. 11. Federation data exchange model – object classes.

8. Evaluation

In this section we evaluate the *S-IDE* tool and discuss the feasibility of the generated deployment model, and the time performance for generating the deployment alternatives.

8.1. Feasibility of the generated deployment model

For analyzing the feasibility of the generated deployment model alternative we use two different approaches.

The first approach is an informal and practical approach based on visual inspection of the generated deployment alternative by an expert. This approach thus relies on the assumption that an expert can provide logical reasoning about the feasibility of the deployment alternative. Note that the generation of the alternative is done automatically and not performed by the expert. An example reasoning of an expert could be based on the deployment alternative given in Fig. 17. A close analysis of this generated deployment alternative shows that the total resource requirements of simulation module instances do not exceed the capacity of the corresponding nodes. 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, the simulation modules *VehicleTracker*, *CarModel*, *TruckModel* and *DriverModel* appeared to have frequent interactions in the publish–subscribe relations model (Fig. 13) and in the simulation execution configuration (Fig. 15) we can observe that they have high update-rates. Likewise, in Fig. 17 the adopted algorithm has co-located instances of these modules as much as possible. The simulation instances that

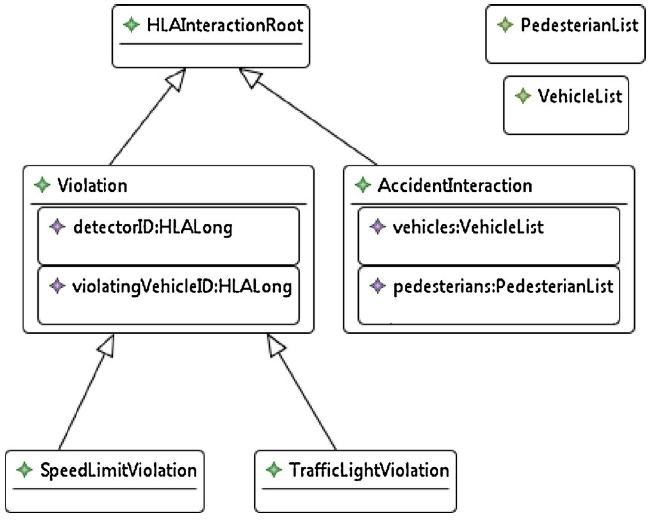


Fig. 12. Federation data exchange model – interaction classes.

are remaining and which would exceed the capacity of Node-1 are deployed to other nodes in a similar manner.

The second, more formal approach for evaluating the generated deployment alternative is to compare the generated alternative with another deployment alternative (Aleti et al., 2009a; Malek et al., 2012). The *S-IDE* tool provides a quality evaluation tool that enables the comparison of two deployment models with respect to given simulation execution configurations. The generated deployment model can be evaluated by comparing it with other deployment models as it was described in Section 4, Step 10.

The comparison process provided in the *S-IDE* is generic and can be applied in a similar way for the alternatives generated with all the three approaches. We show the evaluation of the generated deployment model (Fig. 17) with a manually generated deployment model that is based on the first example expert judgment deployment model given in Section 3.3 (problem statement). We have manually defined the deployment model for the expert judgment deployment alternative in S-IDE environment (Fig. 19). As shown in the figure, all car simulation modules are deployed on the first node, all truck simulation modules are deployed on the second node, all driver simulation modules are deployed on the third node, and the rest of the simulation modules are deployed on the fourth node as it was described in Section 3.3.

The comparison of the automatically generated deployment model alternative (Fig. 17) with the expert judgment deployment alternative (Fig. 19) is given in Table 3. The table shows the execution and communication cost comparisons for each simulation module of the expert judgment deployment alternative and the generated deployment model alternative. The left column includes

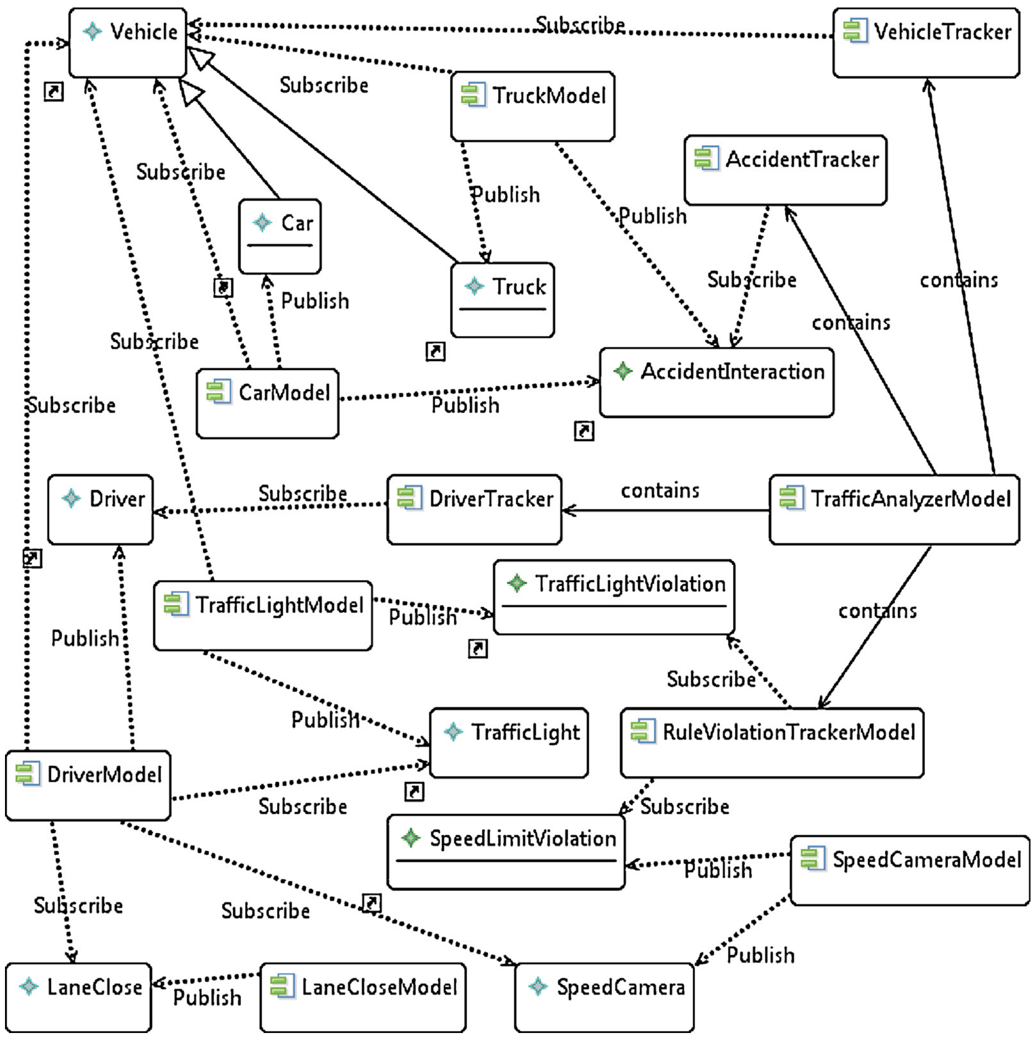


Fig. 13. Module and publish/subscribe definitions for the case study.

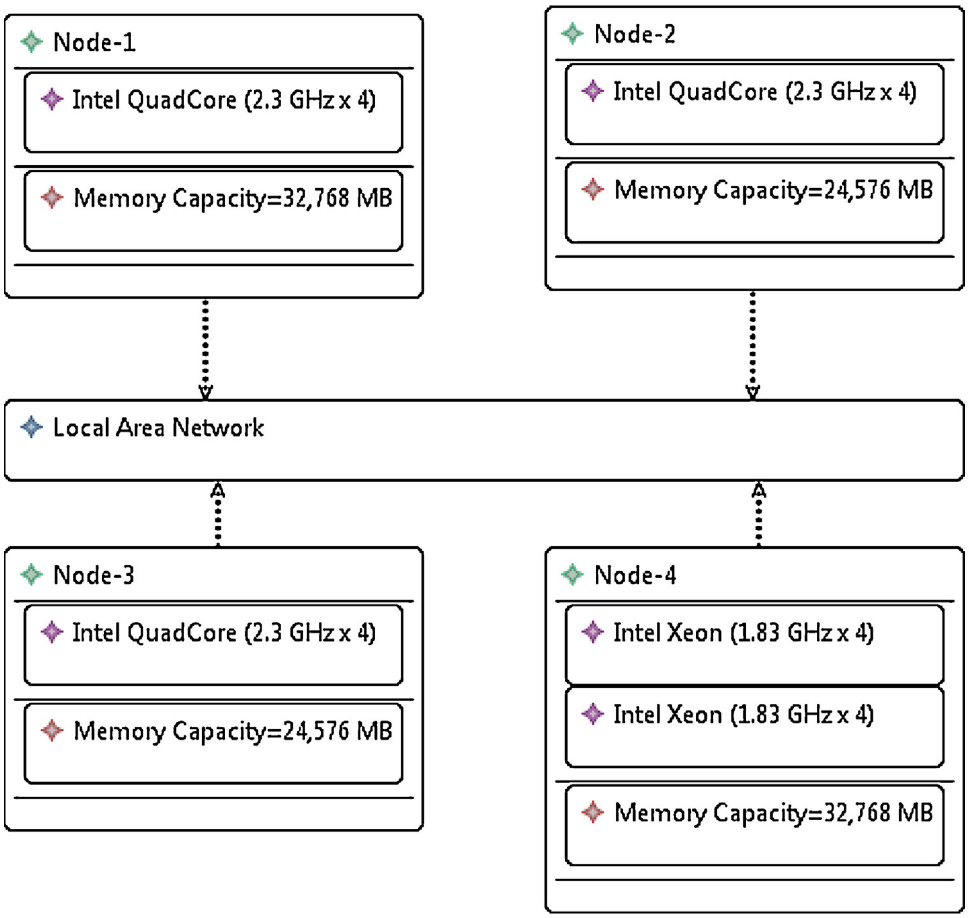

Node-1
Intel QuadCore (2.3 GHz x 4)
Memory Capacity=32,768 MB
Node-2
Intel QuadCore (2.3 GHz x 4)
Memory Capacity=24,576 MB
Local Area Network
Node-3
Intel QuadCore (2.3 GHz x 4)
Memory Capacity=24,576 MB
Node-4
Intel Xeon (1.83 GHz x 4)
Intel Xeon (1.83 GHz x 4)
Memory Capacity=32,768 MB

Fig. 14. A sample physical resource model for the case study with four nodes.

the modules of the deployment alternatives. The total number of each entity in the scenario is shown in parenthesis (e.g. TruckModelInstance (×80) means that there are 80 TruckModuleInstances in the scenario). The column *Execution Cost* defines the values for execution cost for the expert judgment and the generated alternative as well as the improvement percentage of the generated alternative over the manual alternative. Similarly, the column *Communication Cost* defines the values for the communication cost for both alternative models and the improvement percentage. The last row of the table shows the total costs for each deployment alternative and the improvement percentages.

As shown in the table, the total execution cost is optimized by 13.63% in the particular case. It is interesting to see that for some of the simulation modules (e.g. DriverTrackerModelInstance, TrafficAnalyzerModelInstance, etc.) the execution costs seem to be better in the expert judgment deployment alternative. This is because the purpose of the deployment model optimization is to optimize the total performance of the system. For the given case, the modules with the total highest execution cost appeared to be the modules DriverModuleInstances, CarModuleInstances, and TruckModuleInstances with total cost of 4200, 2400 and 400 respectively. For these modules total improvement of 21.94%, 2.88% and 1.72% have been achieved. Although the total execution cost of the other module instances seem to be worse, the impact of the improvement of these three modules seem to define the total improvement in the execution cost.

Table 3
Comparing generated deployment model with manually developed deployment model with expert judgment.

Module	Total execution cost			Total communication cost (MB/second)		
	Expert Judg.	Generated by *S-IDE*	Improv. (%)	Expert Judg.	Generated by *S-IDE*	Improv. (%)
DriverTrackerModelInstance (×1)	5.63	11.25	−100.00	0.0	0.0	0.00
TruckModelInstance (×80)	400	393.13	1.72	6.53	4.90	24.97
SpeedCameraModelInstance (×5)	6.25	8.50	−36.00	0.08	0.06	23.79
TrafficAnalyzerModelInstance (×1)	6.25	12.50	−100.00	0.00	0.00	0.00
CarModelInstance (×600)	2400	2331.00	2.88	29.34	22.00	25.03
RuleViolationTrackerModelInstance (×1)	5.63	9.00	−60.00	0.00	0.00	0.00
VehicleTrackerModelInstance (×1)	5.63	9.00	−60.00	0.00	0.00	0.00
AccidentTrackerModelInstance (×1)	6.25	10.00	−60.00	0.00	0.00	0.00
DriverModelInstance (×680)	4250	3317.50	21.94	0.11	0.08	25.59
LaneCloseModelInstance (×4)	7.50	10.50	−40.00	0.10	0.07	24.15
TrafficLightModelInstance (×15)	18.75	30.50	−62.67	0.18	0.13	25.13
Total costs	7111.88	6142.88	13.63	36.34	27.25	25.02

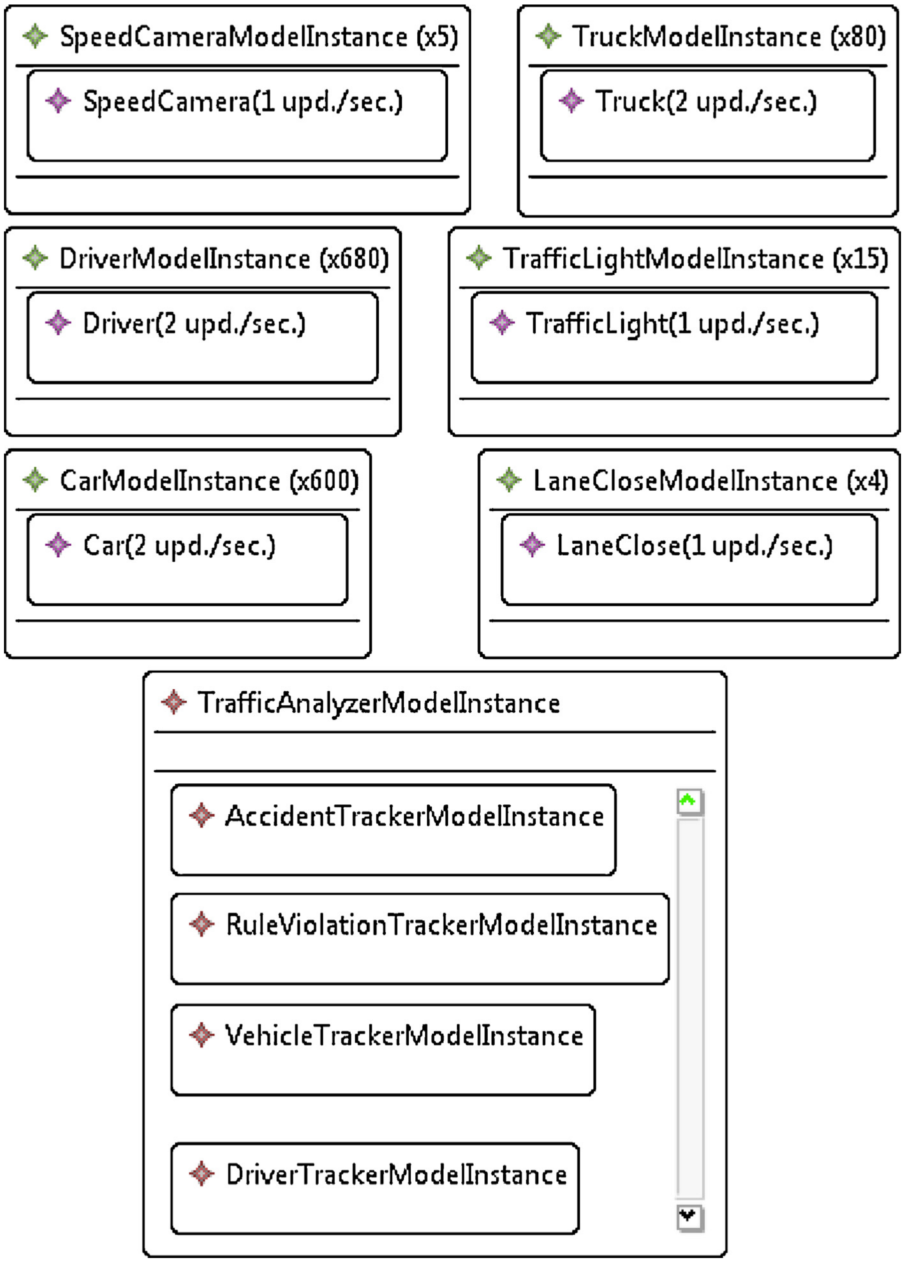

SpeedCameraModelInstance (x5)
SpeedCamera(1 upd./sec.)
TruckModelInstance (x80)
Truck(2 upd./sec.)
DriverModelInstance (x680)
Driver(2 upd./sec.)
TrafficLightModelInstance (x15)
TrafficLight(1 upd./sec.)
CarModelInstance (x600)
Car(2 upd./sec.)
LaneCloseModelInstance (x4)
LaneClose(1 upd./sec.)
TrafficAnalyzerModelInstance
AccidentTrackerModelInstance
RuleViolationTrackerModelInstance
VehicleTrackerModelInstance
DriverTrackerModelInstance

Fig. 15. Simulation execution configuration for the case study.

The total communication cost is optimized by 25.02% for this particular case. Again we can observe that the deployment model is optimized with respect to the total communication performance of the system. As shown in the table, to avoid the duplication of the communication costs, some of the simulation module instances such as *DriverTrackerModelInstance* and *TrafficAnalyzerModelInstance* are not charged any communication costs. For example, the *DriverTrackerModelInstance* subscribes to *Driver* object which is published by *DriverModelInstance* as given in Fig. 13. The cost of this data exchange is only charged to the publisher (*DriverModelInstance*) and is not charged to the subscriber (*DriverTrackerModelInstance*). Since *DriverTrackerModelInstance* does not publish any other object, no communication cost is charged.

We have also carried out the same evaluation for the automatically generated deployment alternative of Fig. 18. The improvement

```
1. GENERATE_FEASIBLE_DEPLOYMENTS (phy_resources, exec_config)
2.        processors ← EXTRACT_PROCESSORS (phy_resources)
3.        tasks ← EXTRACT_TASKS (exec_config)
4.        assignment_tables ← EXECUTE_CTAP (tasks, processors)
5.        CREATE_DEPLOYMENT_MODELS (assignment_tables)
```

Fig. 16. Pseudo-code for generating feasible deployment alternatives.

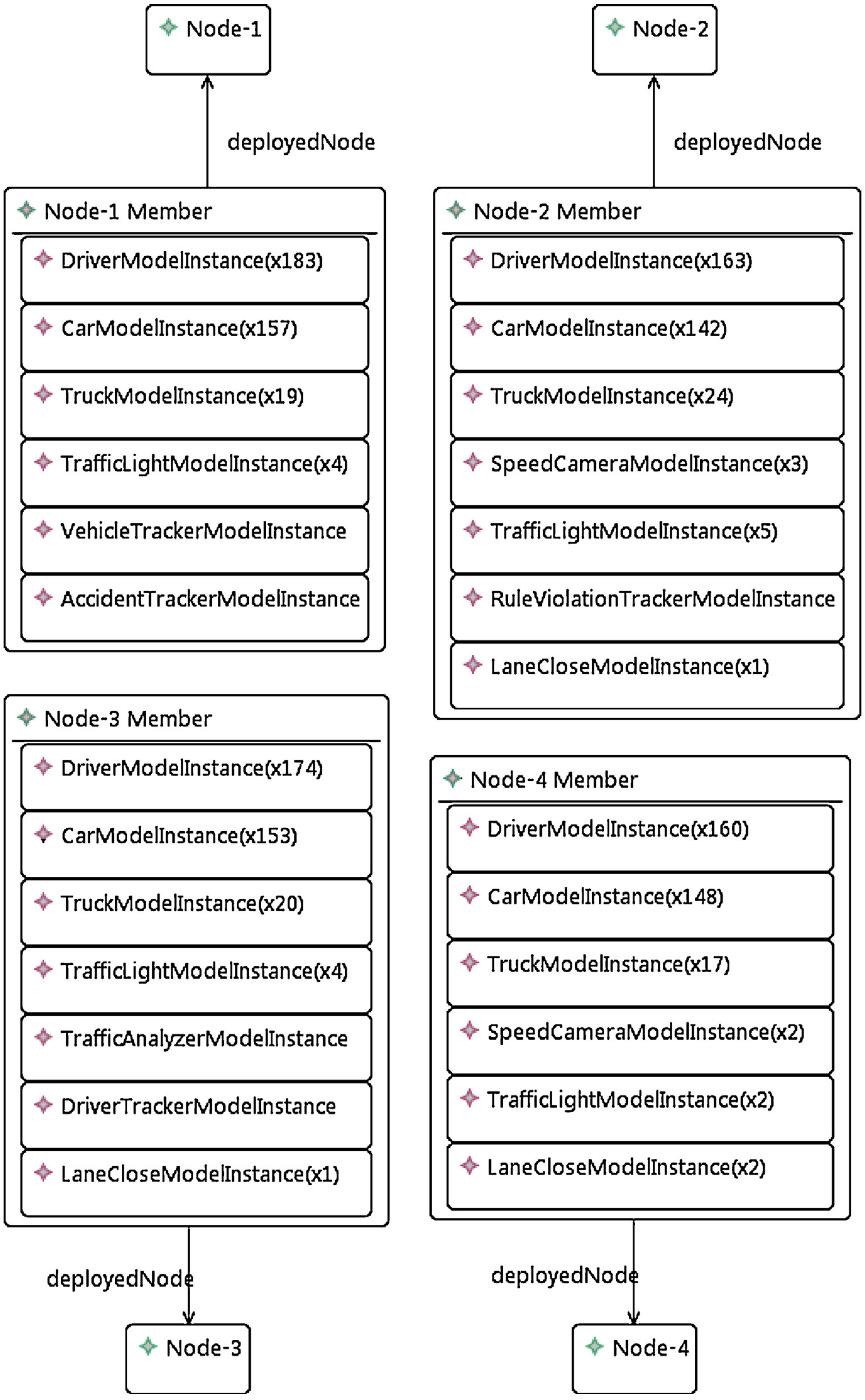

Node-1
deployedNode
Node-1 Member
DriverModelInstance(x183)
CarModelInstance(x157)
TruckModelInstance(x19)
TrafficLightModelInstance(x4)
VehicleTrackerModelInstance
AccidentTrackerModelInstance
Node-2
deployedNode
Node-2 Member
DriverModelInstance(x163)
CarModelInstance(x142)
TruckModelInstance(x24)
SpeedCameraModelInstance(x3)
TrafficLightModelInstance(x5)
RuleViolationTrackerModelInstance
LaneCloseModelInstance(x1)
Node-3 Member
DriverModelInstance(x174)
CarModelInstance(x153)
TruckModelInstance(x20)
TrafficLightModelInstance(x4)
TrafficAnalyzerModelInstance
DriverTrackerModelInstance
LaneCloseModelInstance(x1)
deployedNode
Node-3
Node-4 Member
DriverModelInstance(x160)
CarModelInstance(x148)
TruckModelInstance(x17)
SpeedCameraModelInstance(x2)
TrafficLightModelInstance(x2)
LaneCloseModelInstance(x2)
deployedNode
Node-4

Fig. 17. Sample of generated feasible deployment alternative.

of the total execution cost with respect to the deployment model defined by the expert (Fig. 19) is 14.30%. The improvement of the communication cost appeared to be 24.99%.

We have also compared the two automatically generated deployment alternatives of Figs. 17 and 18. It appears that the second alternative seems to have 0.78% lower execution cost with respect to the first alternative. Further, the first alternative seems to have 0.03% lower communication cost. Based on these results, in this case one would slightly prefer the first alternative if optimizing the communication cost is considered more important than optimizing the execution cost. The second alternative would be selected if execution cost is considered more important. The evaluation of other deployment alternatives can be carried out in a similar manner to find the feasible deployment alternative.

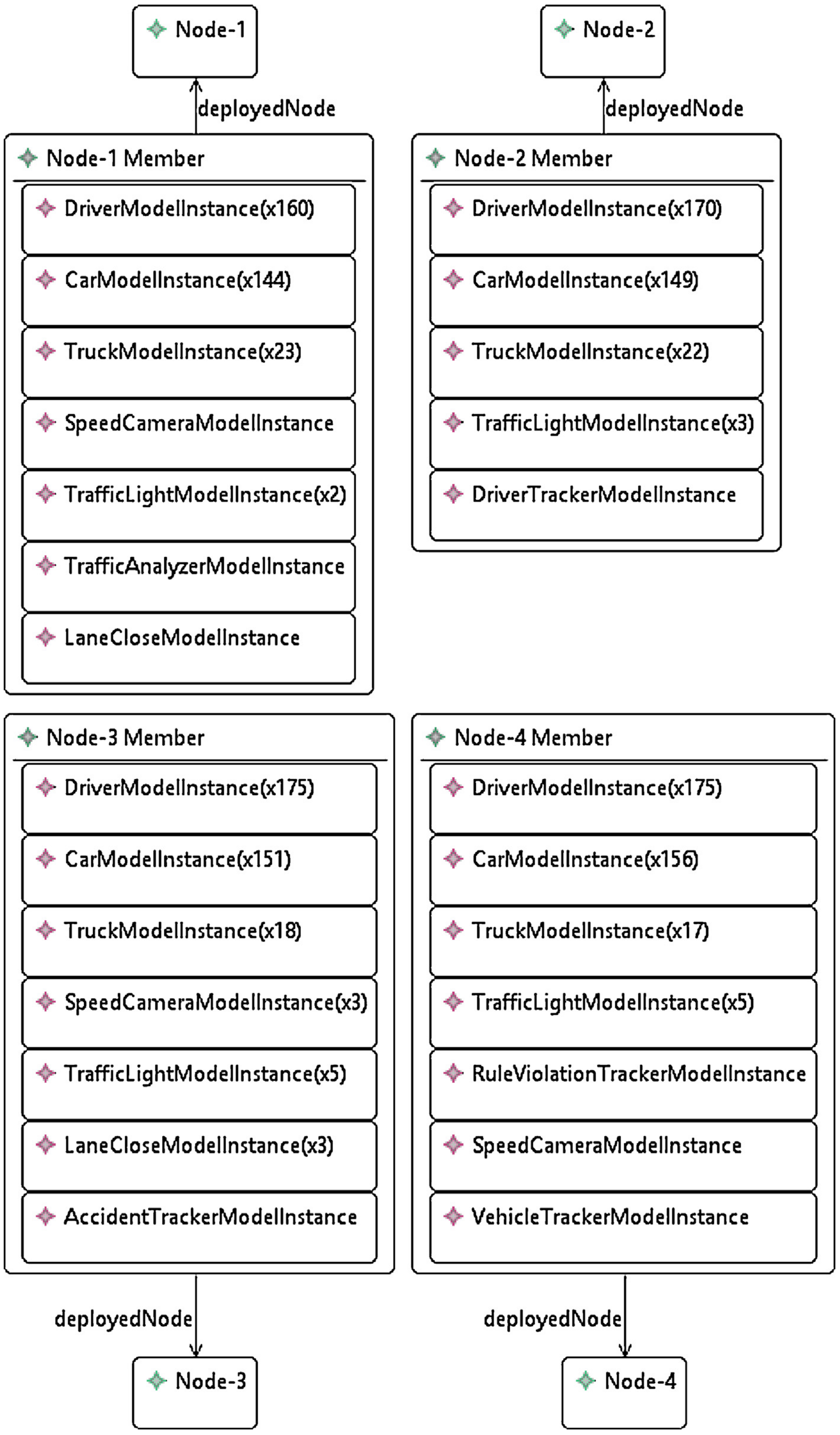

Node-1
deployedNode
Node-1 Member
DriverModelInstance(x160)
CarModelInstance(x144)
TruckModelInstance(x23)
SpeedCameraModelInstance
TrafficLightModelInstance(x2)
TrafficAnalyzerModelInstance
LaneCloseModelInstance
Node-2
deployedNode
Node-2 Member
DriverModelInstance(x170)
CarModelInstance(x149)
TruckModelInstance(x22)
TrafficLightModelInstance(x3)
DriverTrackerModelInstance
Node-3 Member
DriverModelInstance(x175)
CarModelInstance(x151)
TruckModelInstance(x18)
SpeedCameraModelInstance(x3)
TrafficLightModelInstance(x5)
LaneCloseModelInstance(x3)
AccidentTrackerModelInstance
deployedNode
Node-3
Node-4 Member
DriverModelInstance(x175)
CarModelInstance(x156)
TruckModelInstance(x17)
TrafficLightModelInstance(x5)
RuleViolationTrackerModelInstance
SpeedCameraModelInstance
VehicleTrackerModelInstance
deployedNode
Node-4

Fig. 18. Another sample of generated feasible deployment alternative.

8.2. *Deployment model generation performance*

The performance of the deployment model generation process is largely influenced by the performance of the selected CTAP algorithm. For our particular case, the selected generation algorithm (Mehrabi et al., 2009) is implemented in Java and executed on an Intel Core I-7 2.70 GHz 64-Bit computer with 4 GB of RAM. We have used the S-IDE tool to provide four different simulations of the traffic simulation case study. The results of the simulations are shown in Table 4. In addition to the traffic simulation we have also defined four different simulations in the Electronic Warfare (EW) domain (Adamy and David, 2006). Each simulation has been separately defined and executed. Further we have measured the time to generate the deployment alternative for each scenario. We have tried to define simulations which are also realistic from an industrial perspective. From our own industrial experiences in

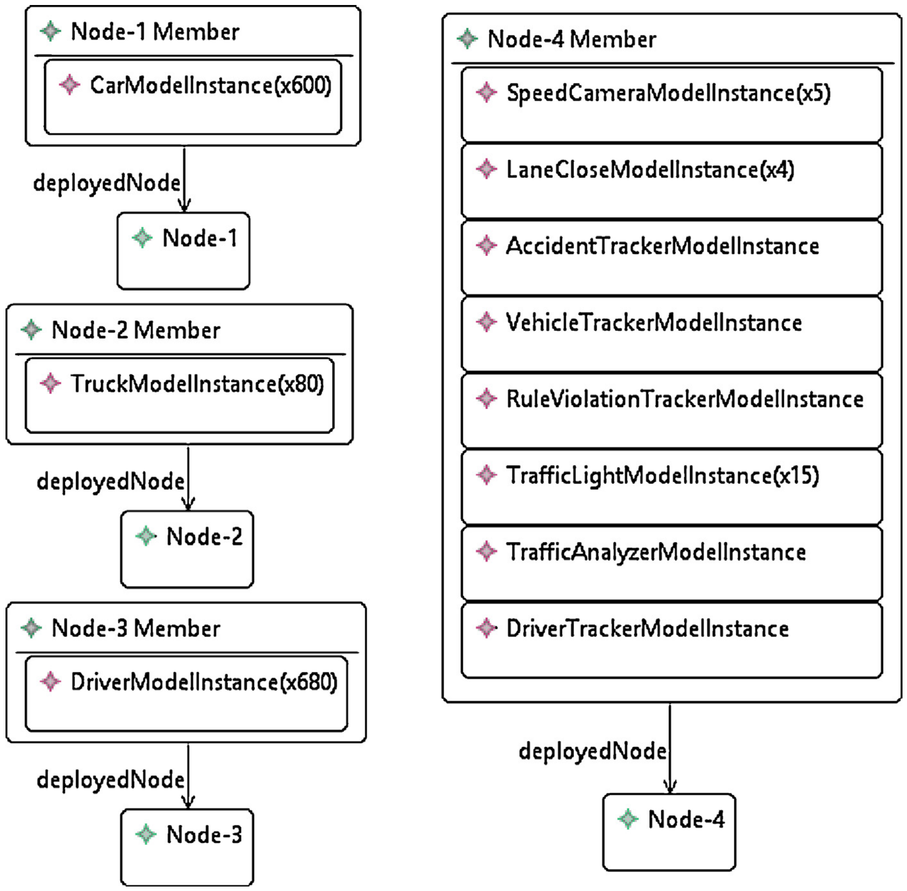

Node-1 Member
CarModelInstance(x600)
deployedNode
Node-1
Node-2 Member
TruckModelInstance(x80)
deployedNode
Node-2
Node-3 Member
DriverModelInstance(x680)
deployedNode
Node-3
Node-4 Member
SpeedCameraModelInstance(x5)
LaneCloseModelInstance(x4)
AccidentTrackerModelInstance
VehicleTrackerModelInstance
RuleViolationTrackerModelInstance
TrafficLightModelInstance(x15)
TrafficAnalyzerModelInstance
DriverTrackerModelInstance
deployedNode
Node-4

Fig. 19. Expert judgment deployment alternative.

the distributed simulation domain we can state that cases from 4 to 12 nodes are realistic, whereby 12 nodes is usually rarely used. We have chosen the simulation examples from 4 to 12 nodes which are all realistic from an industrial perspective. Regarding the number of simulation entities we have used simulations with very low number of entities (the lowest 17) to very high number 1596 entities. Also in this case these examples are realistic in the industrial setting. A medium size of a real scenario in this context is typically a simulation including 4–6 nodes with around 1000 simulation entities.

From Table 4 we can further observe that the generation times of deployment alternatives are acceptable for evaluation at design time. The execution time of the algorithm appears to be dependent on the number of simulation entities and the number of nodes. The higher the number of simulation entities the higher the generation time of the deployment alternative. Further, if the number of nodes increase then finding a feasible deployment alternative will be easier and this will result in a faster generation of the deployment alternative. These observations count for both cases. However, for the traffic case study the generation of the feasible deployment alternatives took longer than for the EW simulations in the simulations with large number of simulation entities. This is due to the different communication patterns and execution cost characteristics. In fact the traffic case study that we have described is even more complex than in the EW domain with respect to communication patterns and execution cost characteristics.

As stated before, for the simulation we have used a particular implementation of the CTAP algorithm. Of course different CTAP algorithm implementations can lead to different generation times. The CTAP algorithm implementation (Mehrabi et al., 2009) that we have selected seems to perform reasonably for design time deployment alternative generation. A further analysis could be performed to identify the best performing algorithm (e.g. Aleti et al., 2009a). The approach that we have presented does not mandate a particular implementation of the CTAP algorithm, and we consider the analysis of the algorithm implementations beyond the scope of this work.

Table 4
Generation time values for scenarios using an implementation of CTAP algorithm.

Simulation no.	Simulation	Number of simulation entities	Number of nodes	Generation time (s)
1.	Traffic simulation	17	4	2
2.	Traffic simulation	81	4	8
3.	Traffic simulation	1389	4	12,115
4.	Traffic simulation	1389	12	4273
5.	EW simulation	17	4	50
6.	EW simulation	81	4	182
7.	EW simulation	141	5	325
8.	EW simulation	1596	6	360

9. Discussion

In this work we have provided a tool framework for deriving a feasible deployment alternative based on the simulation system and the available physical resources for HLA based distributed simulations. The tool framework assists the designer to design the simulation system and derive a feasible deployment model in early system design phase. The necessity and practical value of deriving a feasible deployment model in the design phase is based on the alternative design evaluation related recommendations defined by the IEEE standard FEDEP.

A valid question in this context is whether the adopted algorithm in the tool framework leads to a solution and whether this solution is optimal. We have designed the tool framework to enable the utilization of different CTAP solver algorithms which have been widely addressed in the literature. The tool framework 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 papers and based on this we can assume that a good feasible solution can be derived. In addition, depending on the state of the system, different CTAP solver algorithms may be used to derive a feasible deployment alternative.

As stated before in Section 6.3, the S-IDE tool can adopt different CTAP Solver algorithm implementations through OSGI service registry. Since different simulation contexts might require different CTAP Solver implementations, the selection of the appropriate algorithm implementation is left to the designer's decision. To support the designer in selecting the algorithm implementation, the S-IDE tool provides the objectives and characteristics of the algorithm that are defined by the algorithm developer. If needed, the designer can use multiple different algorithm implementations and compare the simulation results with the S-IDE design evaluation tools (see Section 8.1), to select the most appropriate one.

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 (see Section 6.4). The *S-IDE* tool also provides design diagnostic tools that enable the analysis of the simulation design to detect potential bottlenecks such as big sized data exchange objects, high communication costs, limited physical resources and high memory requirements of the simulation modules.

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 3 (problem statement), deferring the definition of the deployment to the development phase might lead to non-feasible 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 FEDEP 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 derived 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 run-time parameter values can be used, instead of estimated values, to derive the deployment model. The runtime parameter values can be collected using HLA Management Object Model (MOM) services as defined in IEEE (2010b).

The *S-IDE* tool framework can be used for design level analysis including, the impact of adding new simulations modules to the system, suitability of the selected physical resources for the given simulation design, and the impact of the change of publish–subscribe relations. To perform such analyses, the designer can define different alternative design models, generate deployment models, and compare quality of the generated deployment models as defined in Section 8.1. As such, the designer can observe the effect of design variations on the performance of each simulation module and the overall system.

The primary goal of our analysis is to find feasible deployment alternatives given the simulation execution configurations, which is based on user-defined scenarios. We have chosen for this approach because in FEDEP also first scenarios are defined and based on these the design models are derived. In fact the output of the tool framework can be further detailed, by addressing, for example, maximum number of simulation module instances that can be deployed on the available physical resources, the upper bound for the update rate of the simulation module instances, the minimum processor frequency and/or memory capacity that is necessary for the defined simulation module instances, etc. However, these analysis would require executing the corresponding CTAP Solver algorithm many times which would be less tractable and as such less practical. This is because in the simulation domain that we have focused on, the real life scenarios can contain thousands of entities and up to dozens of nodes. Nevertheless, if needed the designer can execute the deployment model generation tool several times to find out the boundary values.

In this work the *S-IDE* tool framework is used for deriving feasible deployment alternatives for HLA based simulation systems. However, the tool framework can also be adapted for other architectures that adopt publish–subscribe model, such as OMG DDS (OMG, 2006) or TENA (Noseworthy, 2008). In this case, the metamodels and tools will be modified 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.

In the industrial context we have applied our approach to derive feasible deployment alternatives for an Electronic Warfare (EW) simulation. The EW simulation scenarios include around 4–8 nodes and about 1600 simulation entities. We have deliberately not chosen this domain as a running example in this paper because understanding the EW case study requires more effort than the more conceptual domain of traffic simulation. Yet, the simulations that we have selected for the traffic domain are at least as complex as that of the EW simulation domain. In this respective, the simulations that we have carried are realistic. The scalability and output quality of the approach have been analyzed with respect to the number and characteristics of simulation entities, the number of nodes and the adopted implementation of the algorithm in Section 8.

10. Related work

In this paper we have provided a model-driven development approach for generating and evaluating deployment alternatives for HLA-based simulation systems. The adoption of model-driven

Table 5
Related metamodel based model-driven HLA tools in the literature.

Related work	Description	Supported FEDEP steps	Adopted metamodels	Model transformations	Supports generation of deployment alternatives
KAMA (Karagöz and Demirörs, 2007)	Provides a mission space conceptual modeling environment.	Step 1, Step 2	KAMA metamodel	No	No
BOM (Base Object Model) (SISO, 2006)	Provides a simulation space conceptual modeling environment.	Step 2	BOM metamodel	No	No
Federation Architecture Metamodel-FAMM (Topçu et al., 2008)	Provides a metamodel – federation architecture metamodel (FAMM) – and tool support for designing HLA federates and federations.	Step 3	FAMM	No	No
FAMM based code generation tool (Adak et al., 2010)	Provides a code generation tool based on the FAMM defined above. Generates application code from FAMs (Federation Architecture Model) that are instances of FAMM.	Step 4	FAMM	FAM to application code generation (Java)	No
HLA Object Model Development Tool (Çetinkaya and Oguztüzün, 2006)	Provides tool support for designing Federation Object Models (FOM) and Simulation Object Models (SOM) in compliance with HLA OMT standard.	Step 3	HLA OMT	No	No
A visual tool to simplify the building of distributed simulations using HLA (Parr, 2003)	This tool provides a design and automatic code generation environment for HLA based distributed simulations.	Step 3, Step 4	HLA OMT UML metamodel	Simulation model (PIM) > specialized simulation model (PSM) > application code	No
An HLA-based tactical environment application framework (Çelik et al., 2012)	Provides tools for designing HLA Federation Object Models (FOM) and Simulation Object Models (SOM), defining federates, matching FOM elements with federates, automatic code generation for FOM elements, a scalable and high performance simulation engine that enables execution of multiple simulation module instances in one federate, test, debugging and data record/replay tools.	Step 3, Step 4, Step 5, Step 6, Step 7	HLA OMT	HLA OMT to Programming Language Classes Code Generation (Java)	No
S-IDE tool framework	Described in this paper.	Step 3	Defined in Section 5	Defined in Section 8	Yes

development approaches for simulations systems has also been promoted by other authors. Early on, Tolk has indicated that model-driven architecture needs to be applied to HLA for providing tool support and efficient development (Tolk, 2002).

To support model-driven development, various metamodels and approaches have been proposed in the literature for modeling HLA based simulations. The 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 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 (Parr, 2003; Çetinkaya and Oguztüzün, 2006). The Federation Data Exchange Metamodel that we have defined in this paper extends the HLA OMT for supporting the derivation of feasible deployment alternatives. Based on the conceptual models that are developed using the metamodels such as BOM, HLA based simulation systems can be designed using the UML (Fowler, 2003) or UML profiles (Guiffard et al., 2006; Çelik, 2005). Topçu et al. (2008) adopt Live Sequence Charts (Brill et al., 2004) to model the behavior of HLA federates in simulation systems. Each of these tools provide support for different aspects of HLA-based simulation development domain. However, no adequate and explicit support has been provided for selecting and evaluating the deployment alternatives. In Table 5 we provide the list of these related metamodel-based model-driven HLA tool support approaches and provide a characterization for each tool. The last row of the table includes the characterization of the S-IDE tool. The different tool approaches are compared with respect to the *description, supported FEDEP steps, adopted metamodels, adopted model transformations* and *support for generation of deployment model alternatives* fields. *Supported FEDEP steps* field maps each work with the related FEDEP steps (IEEE, 2003) including "Step-1 Define federation objectives", "Step-2 Perform conceptual analysis", "Step-3 Design federation", "Step-4 Develop federation", "Step-5 Plan, integrate, and test federation", "Step-6 Execute federation and prepare outputs", "Step-7 Analyze data and evaluate results".

In this paper we have defined an approach and tool framework for optimizing deployment architectures of HLA based simulation systems. Related to our work there are several other approaches in other domains for optimizing deployment architectures. The general motivation in these approaches is similar to our motivation for defining a formal method for optimizing deployment architectures. In this context, Kugele et al. (2008) define an approach for optimizing deployment model of embedded systems by using non-functional requirement annotations. The authors focus on the non-functional requirements for *Computing Power, Memory*, and *Power State*. The computing power and memory requirements map to our *Processing Power* and *Memory Requirement* parameters. The *Power State* requirement is defined because of the limited power supplies of embedded systems. Since our target environment is not embedded systems, this requirement is not applicable to our approach. Similar to our approach, Kugele et al. (2008) convert the problem to an optimization problem. Hereby, the necessary inputs of the optimization problem are extracted from non-functional requirements while we extract the inputs from the simulation design model. Further, the authors adopt an integer linear programming (ILP) approach for solving the optimization problem while we do not mandate any approach but use a genetic algorithm based heuristic approach as a sample realization.

Zheng et al. (2007) define an approach to optimize the task placement and the signal to message mapping in a hard real-time distributed environment. The method is applied to an automotive case study. The problem is expressed as an optimization problem to minimize the sum of latencies by finding best (1) task-CPU assignment, (2) signal-message packing, (3) task and message priorities considering constraints on end-to-end signal latencies and message size. Zheng et al. (2007) used Mixed Integer Linear Programming (MILP) techniques and used CPLEX (IBM, 2010) as MILP solver.

Aleti et al. (2009a) discuss the adoption of constructive algorithms instead of iterative evolutionary algorithms for deployment optimization of embedded systems. Constructive algorithms often converge quickly and produce diverse solutions when compared to iterative algorithms (Blum and Roli, 2003). Aleti et al. (2009a) adapt Pareto-Ant Colony Optimization (P-ACO) algorithm (Doerner et al., 2004) to solve a multi-objective deployment optimization problem. The performance of P-ACO is compared with a Multi-Objective Genetic Algorithm (MOGA) by using the Archeopterix tool platform (Aleti et al., 2009b). The parameters for the optimization problem are *memory requirement*, *communication frequencies, and event sizes for components (tasks), memory capacity, network bandwidth, network delay for hosts (processors).* Different from our approach, this problem definition does not define parameters for *processing power*, but includes additional *network bandwidth* and *network delay* parameters.

Malek et al. (2012) propose an extensible framework (*Deployment improvement framework – DIF)* for improving deployment architecture of distributed systems. The authors propose a generic approach that can work with user defined Quality of Service (QoS) dimensions such as latency, security and availability. The proposed framework realizes four different multidimensional optimization problem solving techniques, and provides several novel heuristics for improving the performance of these techniques. Malek et al. (2012) propose generic QoS dimensions while QoS dimensions in our problem are fixed (*communication and execution costs*). Further, the authors mention that the largest scenarios they have worked with to date have involved hundreds of software components and system services. Due to the nature of the simulation domain, we have to work with thousands of entities.

11. Conclusion

One of the important problems in HLA based distributed simulation systems is the allocation of the different simulation modules to the available physical resources. Usually, the deployment of the simulation modules to the physical resources can be done in many different ways. We have defined a method for automatically deriving deployment alternatives. Obviously the method would not be feasible without adequate tool support. We have provided a tool framework, *S-IDE* (Simulation-IDE) that provides an integrated development environment for deriving a feasible deployment alternative based on the simulation system and the available physical resources at the design phase. We have defined several tools in context of the tool framework including *Federation Data Exchange Model Design Tool*, *Simulation Modules and Publish–Subscribe Relations Design Tool, Physical Resources Design Tool*, and *Simulation Execution Configuration Design Tool.* Based on the design models developed with these tools, the necessary parameter values for the CTAP algorithm have been defined for automatic generation of a feasible deployment alternative. To illustrate the usage of the *S-IDE* framework we have adopted a realistic case study concerning the development of a traffic simulation. The generation times of the deployment alternatives were acceptable for evaluation at design time. 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 tool framework 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 FDEM to the target platform. Further, we aim to integrate behavioral modeling to consider also dynamic aspects of HLA based simulation systems. We will also work on adopting different optimization techniques in the S-IDE framework. In this context, we think that adopting a constructive algorithm like P-ACO (Aleti et al., 2009a) and comparing the results with our out-of-the-box Genetic Algorithm (Mehrabi et al., 2009) will be interesting. Another interesting work would be to define our specific QoS parameters and sample scenarios in DIF (Malek et al., 2012) and use its output to generate deployment model in S-IDE.

References

Adak, M., Topçu, O., Oguztüzün, H., 2010. Model-based code generation for HLA federates. Software: Practice and Experience 40 (2), 149–175.

Adamy, David, L., 2006. Introduction to Electronic Warfare Modeling and Simulation. Artech House Inc., Norwood MA, USA.

Aleti, A., Grunske, L., Meedeniya, I., Moser, I., 2009a. Let the ants deploy your software—an ACO based deployment optimisation strategy. In: ASE'09 Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 505–509.

Aleti, A., Bjornander, S., Grunske, L., Meedeniya, I.,2009b. Acheopterix: An extendable tool for architecture optimisation of AADL models. In: MOMPES'09. IEEE Digital Libraries, pp. 61–71.

ATL, 2012. Eclipse Atlas Transformation Language Project. Eclipse Foundation http://www.eclipse.org/atl/

Bezivin, J., 2005. On the unification power of models. Software and System Modeling 4, 171–188.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys 35 (3), 268–308.

Brill, M., Damm, W., Klose, J., Westphal, B., 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 3147, 374–399.

Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Timothy, J.Grose, 2003. Eclipse Modeling Framework. Addison-Wesley Professional, Boston, MA, USA.

Büttner, F., Cabot, J., Gogolla, M., 2011. On validation of ATL transformation rules by transformation models. In: Stephan Weißleder, Levi Lúcio, Harald Cichos, Frédéric Fondement (Eds.), Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa). ACM, New York, NY, USA, Article 9, 8pp.

Chung, Y.S., Wong, J.T., 2010. Investigating driving styles and their connections to speeding and accident experience. Journal of the Eastern Asia Society for Transportation Studies, 8.

Czarnecki, K., Helsen, S., 2006. Feature-based survey of model transformation approaches. IBM Systems Journal 45 (3), 621–645.

ÇELIK, T., 2005. A Software Modeling Tool for the High Level Architecture, Master's Thesis, Hacettepe University Institute of Science.

Çelik, T., Gökdoğan, F.G., Öztürk, K., Sarikaya, B., 2012. An HLA-based tactical environment application framework. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 23 (November), doi:1548512912465993.

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

Daly, C., 2004. Emfatic Language Reference. http://www.eclipse.org/gmt/epsilon/doc/articles/emfatic/

Doerner, K., Gutjahr, W.J., Hartl, R.F., Strauss, C., Stummer, C., 2004. Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection. Annals of Operations Research 131 (1–4), 79–99.

Equinox, 2012. Equinox OSGI Project. Eclipse Foundation http://www.eclipse.org/equinox

Eugster, P.T.H., Felber, P.A., Guerraoui, R., Kermarrec, A., 2003. The many faces of publish/subscribe. ACM Computing Surveys 35 (2), 114–131.

Fowler, M., 2003. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Frankel, D.S., Parodi, J., Soley, R., 2004. The MDA Journal: Model Driven Architecture Straight From The Masters. Meghan-Kiffer Press, Tampa, FL, USA.

Fujimoto, R.M., 1999. Parallel and Distributed Simulation Systems, 1st ed. John Wiley & Sons, Inc., New York, NY, USA.

Gra2Mol, 2012. Grammar to Model Language Project. Modelum Research Team, University of Murcia http://modelum.es/trac/gra2mol/

Gronback, R.C., 2009. Eclipse Modeling Project: A Domain-specific Language Toolkit. Addison-Wesley Professional, Upper Saddle River, NJ, USA.

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

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.

Izosimov, V., Pop, P., Eles, P., Peng, Z., 2005. Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems. In: Design, Automation and Test in Europe, 2005 Proceedings, vol. 2, 7–11 March, pp. 864–869.

Izquierdo, J.L.C., Molina, J.G., 2009. A domain specific language for extracting models in software modernization. In: Proceedings of the 5th European Conference on Model Driven Architecture – Foundations and Applications, ECMDA-FA'09. Springer-Verlag, Berlin, Heidelberg, pp. 82–97.

JET, 2012. Eclipse Java Emitter Transformations Project. Eclipse Foundation http://www.eclipse.org/modeling/m2t/?project=jet#jet

JUNIT, 2012, JUnit Project Home Page, http://www.junit.org/

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

Kolovos, D.S., Paige, R.F., Polack, F.A.C., 2006. Eclipse Development Tools for Epsilon. In: Eclipse Summit Europe, Eclipse Modeling Symposium.

Kolovos, D.S., Rose, L.M., Abid, S., Paige, R.F., Polack, F.A.C., Botterweck, G.,2010. Taming EMF and GMF using model transformation. In: Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 6394, chapter 15. Springer, Berlin/Heidelberg, pp. 211–225.

Kugele, S., Haberl, W., Tautschnig, M., Wechs, M., 2008. Optimizing automatic deployment using non-functional requirement annotations. In: Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA.

Kuhl, F., Weatherly, R., Dahmann, J., 1999. Creating Computer Simulation Systems: An Introduction to the High Level Architecture, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA.

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

Lo, V.M., 1988. Heuristic algorithms for task assignment in distributed systems. IEEE Transactions on Computers 37 (11), 1384–1397.

Malek, S., Medvidovic, N., Mikic-Rakic, M., 2012. An extensible framework for improving a distributed software system's deployment architecture. IEEE Transactions on Software Engineering 38 (1), 73–100.

McAffer, J., Vanderlei, P., Archer, S., 2010. Osgi and Equinox: Creating Highly Modular Java Systems., 1st ed. Addison-Wesley Professional, Upper Saddle River, NJ, USA.

Mehrabi, A., Mehrabi, S., Mehrabi, A.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 2009, vol. II.

Moore, W., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P., 2004. Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Corp., Riverton, NJ, USA.

Noseworthy, J.R., 2008. The Test and Training Enabling Architecture (TENA) Supporting the Decentralized Development of Distributed Applications and LVC Simulations. In: 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, Vancouver, BC, Canada, pp. 259–268.

Parr, S., 2003. A visual tool to simplify the building of distributed simulations using HLA. Information & Security Journal 12 (2), 151–163.

Pirim, T., 2006. A hybrid metaheuristic algorithm for solving capacitated task allocation problems as modified XQX problems. Ph.D. Dissertation. University of Mississippi, MS, USA. Advisor(s) Bahram Alidaee, AAI3259418.

Podgorelec, V., Hericko, M., 2007. Estimating software complexity from UML models. ACM SIGSOFT Software Engineering Notes 32 (2), 1–5.

OMG, 2006. Data Distribution Service for Real-time Systems Ver 1.2.

OSGI, 2011. OSGI Service Platform Core Specification Release 4, Version 4.3. OSGI Alliance.

QVT, 2012. Eclipse Query/View/Transformations Project. Eclipse Foundation http://www.eclipse.org/m2m/

Schmidt, D.C., 2006. Model-driven engineering. IEEE Computer 39 (2), 25–32.

SIDE, 2012. SIDE Tool Framework Project. Hacettepe University http://web.cs.hacettepe.edu.tr/~turgay/SIDE

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 Transactions on Software Engineering 3 (1), 85–93.

Taubman-Ben-Ari, O., Mikulincer, M., Gillath, O., 2004. The multidimensional driving style inventory—scale construct and validation. Accident Analysis and Prevention 36 (3), 323–332.

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

Topçu, O., Adak, M., Oguztüzün, H., 2008. A metamodel for federation architectures. ACM Transactions on Modeling and Computer Simulation 18 (3), 1–29.

Voelter, M., Kolb, B., Efftinge, S., Haase, A., 2006. From Front End to Code—MDSD in Practice. http://www.eclipse.org/articles/Article-FromFrontendToCode-MDSDInPractice/article.html

Xpand, 2012. Eclipse XPand Project. Eclipse Foundation http://www.eclipse.org/modeling/m2t/?project=xpand

Xtext, 2012. Eclipse XText Project. Eclipse Foundation http://www.eclipse.org/Xtext

Zeigler, B.P., 2003. DEVS today: recent advances in discrete event-based information technology. In: 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, MASCOTS, pp. 148–161.

White, J., Schmidt, D.C.,2010. R&D challenges and emerging solutions for multicore deployment/configuration optimization. In: Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER'10). ACM, New York, NY, USA, pp. 407–410.

Zheng, W., Zhu, Q., Natale, M.Di, Vincentelli, A.S.,2007. Definition of task allocation and priority assignment in hard real-time distributed systems. In: Proceedings of the 28th IEEE International Real-time Systems Symposium (RTSS'07). IEEE Computer Society, Washington, DC, USA, pp. 161–170.

Turgay Çelik received his BS (2003), MSc (2005) and PhD (2013) degrees in Computer Engineering from the Hacettepe University, Turkey. From 2003 to 2005 he served as a research assistant in Hacettepe University. Currently, he is a Lead Software Engineer in MilSOFT Inc. Turkey, where he has been working since 2005. He has 10 years of professional experience in software engineering research and software development. His research topics include distributed systems, infrastructure and middleware technologies, modeling and simulation, software architecture modeling, model-driven software development, software design optimization, and software performance profiling and optimization.

Bedir Tekinerdogan received his MSc degree in Computer Science in 1994, and a PhD degree in Computer Science in 2000, both from the University of Twente, The Netherlands. From September 2003 until September 2008 he served as an assistant professor at University of Twente. Currently he is an assistant professor at Bilkent University in Turkey where he is leading the Bilkent Software Engineering Group. He has around 20 years of professional experience in software engineering research and education. His key research topic is software architecture design and related to this model-driven software development, software product line engineering, global software development, and aspect-oriented software development.