International Journal of
Computer & Software Engineering

Research Article Open Access

Special Issue: Software Architecture

Architecting Feasible Deployment Alternatives for Publish-Subscribe Systems

Bedir Tekinerdogan[1]* and Turgay Celik[2]
[1]Information Technology, Wageningen University, Wageningen, The Netherlands
[2]Department of Computer Engineering, HacettepeUniversity, Ankara, Turkey

Abstract

Publish-Subscribe is one of the important patterns for developing scalable distributed systems. Usually, the deployment of the publishers and subscribers to the nodes can be done in many different ways, whereby each deployment alternative will have a different impact on the performance. The many possible architecture design alternatives tend to trade-off with respect to execution cost and communication cost. Unfortunately, for the human engineer it is not tractable to define a feasible configuration in case of large number of nodes and participants. In this paper we propose a generic method to assist the architect by automatically deriving feasible deployment alternatives of Publish-Subscribe based distributed systems. The approach is based on the so-called capacitated task allocation problem (CTAP) in which constraints on memory capacity and processing power are applied to manage the trade-offs between the total execution cost and total communication cost to derive feasible design alternatives. The method is supported by our tool framework (*Deploy-PS*) that provides an integrated development environment for modeling the Publish-Subscribe deployment architecture, modeling the physical resources and the performance requirements, and the selection and generation of the feasible deployment architecture alternatives.

Publication History:

Received: March 28, 2017
Accepted: August 14, 2017
Published: August 16, 2017

Keywords:

Publish-subscribe architecture, Model-driven development, Architectural trade-Off analysis, Tool support

Introduction

A distributed system consists of multiple software components that are located on networked computers, but act and run as a single system. The computers that are in a distributed system can be connected by a local network and be physically close to each other, or they can be connected in a wide area network and geographically distant. Distributed systems offer many benefits over centralized systems, including scalability, concurrency and redundancy. The components in a distributed system communicate and coordinate their actions by passing messages to achieve a common goal. There are many alternatives for the message passing mechanism in distributed systems, including the request-reply pattern and *publish-subscribe* pattern [1]. The publish-subscribe pattern has gained broad attention in the development of loosely coupled, scalable large-scale applications. In distributed systems with the publish/subscribe interaction pattern, so-called *subscribers* express their interest in an event, or a pattern of events, and are subsequently asynchronously notified of events generated by *publishers*.

The publish-subscribe pattern [1] has been used in several different standard infra-structures such as the Java Message Service (JMS) [2], Data Distribution Service (DDS) [3], Distributed Interactive Simulation (DIS) [4] and High Level Architecture (HLA)[5]. These infrastructures help to reduce the effort for developing publish-subscribe systems but do not impose constraints on the deployment of the compo-nents to the different nodes. Given the large number of components involved in distri-buted systems, usually many different deployment configuration alternatives are possible that tend to trade-off with respect to execution cost and communication cost. Unfortunately, for the human engineer it is not tractable to define a feasible configuration in case of large number of nodes and participants.

The deployment of participants to nodes can be generalized to the so-callled *task allocation problem* that has been widely addressed in the literature [6,7]. It is generally known that the solution of the task allocation problem can quickly lead to a design space that is not tractable for the human designer. Hence, the evaluation of the deployment alternative is usually based on expert judgment and postponed to the implementation phase. Relying on expert judgment for defining the feasible deployment of the components, however, is limited, since designing deployment model of the system requires knowledge on underlying technology and the application domain. It is not always possible to find experts that have both knowledge on the corresponding domain and the technology infrastructure. Postponing the design decisions to the implementation might easily lead to an improper configuration with respect to performance requirements.

In this paper we propose a generic method for systematically selecting and generating deployment alternatives for Publish-Subscribe based distributed systems. The approach is an abstraction of our earlier work on defining feasible configuration alternatives in HLA based distributed simulation systems [8,9]. The approach that we present in this paper is generic and can be applied to a broader set of publish subscribe systems, beyond simulation systems. We use the so-called capacitated task allocation problem (CTAP) in which constraints on memory capacity and processing power are applied to manage the trade-offs between the total execution cost and total communication cost to derive feasible design alternatives. We have developed a tool framework (Deploy-PS) that provides an integrated development environment for deriving feasible deployment

***Corresponding Author:** Dr. Bedir Tekinerdogan, Information Technology, Wageningen University, Wageningen, The Netherlands; E-mail: bedir.tekinerdogan@wur.nl

Citation: Tekinerdogan B, Celik T (2017) Architecting Feasible Deployment Alternatives for Publish-Subscribe Systems. Int J Comput Softw Eng 2: 117. doi: https://doi.org/10.15344/2456-4451/2017/117

alternatives based on the application and the available physical resources at the design phase. The method and the tool support have been validated by using two different case studies for the development of a traffic simulation system [8], an electronic warfare simulation [9], and a DDS-based city wide Advanced Traffic Management System (ATMS) [10].

The remainder of the paper is organized as follows. In section 2 we provide the background to support the understanding of the approach. Section 3 describes the problem statement in more detail. Section 4 presents the approach for evaluating alternative design options briefly. Section 5 describes the generic metamodel that will be specialized for different Publish-Subscribe systems. Section 6 presents the corresponding tool support. Section 7 provides the evaluation. Section 8 describes the related work and finally section 9 concludes the paper.

Publish-Subscribe Architectures

As stated before the publish-subscribe interaction pattern has been applied in several applications and infrastructures, which share similar structure and concepts. Figure 1 shows the result of a domain analysis to publish-subscribe systems and represents the reference architecture of these systems.

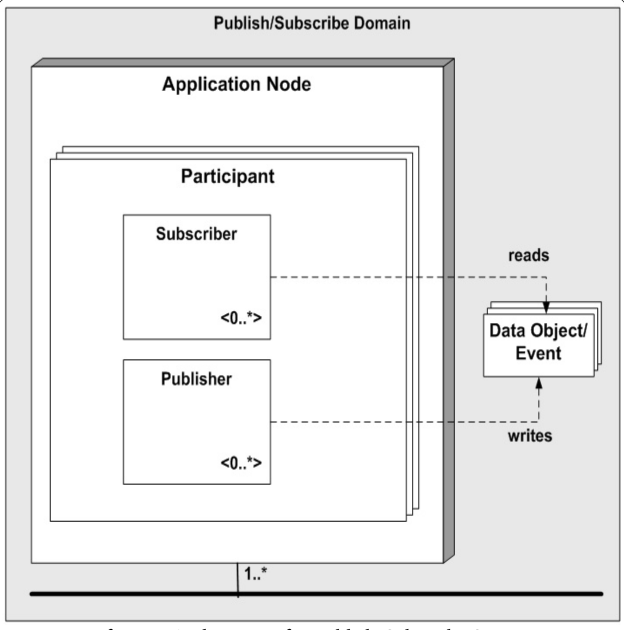


Figure 1: Reference Architecture for Publish-Subscribe Systems.

A typical Publish-Subscribe system defines a *Publish-Subscribe Domain* which consists of a group of Participants which are deployed on a number of *Application-Nodes*. Each Participant defines a number of *Publisher* and *Subscribers* that reads/writes *Data Objects/Events*. *Data Objects/Events* are elements of data exchange model of the publish-subscribe system. Three different types of decoupling can be identified between the subscribers and publisher [1]. *Time decoupling* refers to the fact that interacting components do not need to be actively participating in the interaction at the same time. Publishers might publish events independent of the subscribers, and subscribers might get notified about the occurrence of events even if the original publisher of the event is disconnected. *Space decoupling* refers to the fact that publishers and subscribers might not know each other and do not hold any reference to each other. Finally, *synchronization decoupling* refers to the fact that publishers and subscribers are not blocked during their actions. Based on a thorough domain analysis to existing publish-subscribe middleware systems [11-13] we have derived a feature model that is shown in Figure 2. Here the publish-subscribe middleware systems can be distinguished based on the *type* and the *service model.* Regarding the type we can identify data-centric, message-centric or object-centric approaches. In the message-centric approach, the middleware is not aware of the content of the data; it is just responsible for transmitting the messages among participants. In data-centric approach, the middleware is aware of the content and can impose quality of service parameter values on the data. In object-centric approaches the middleware is responsible of transmitting objects among participants. The *service model* of a publish-subscribe middleware can be characterized based on (1) Communications Model, (2) Architecture Model and (3) Object Model. Communication Model defines communication approach that is applied by the participants. The communication approach on its turn can be based on data distribution, shared data, queuing, and remote procedure call. The *Architecture Model* of a middleware can be either centralized or decentralized denoting whether the data flows through a central unit or not. Further, the architecture model can include a broker that manages the data flow. The architecture can be *unbrokered*, i.e. there is no broker defined, or multi-brokered, whereby multiple brokers manage the data flow. The final distinguishing character of the service model is the adopted Object Model that defines the type of middleware entities that is adopted in the interaction among participants.

In the state-of-the-art we can identify several publish-subscribe middleware ap-proaches which are listed in Table 1. The columns of the table describe the features of the feature model of Figure 2. From the table we can observe that publish-subscribe systems like DDS, HLA, DIS, and TENA are based on data distribution in which publishers and subscribers share data with each other directly. In JMS, a queue based approach is used in which messages are pushed to centralized queues and delivered to the consumers. CORBA is an approach in which by default remote procedure call approach is used. Note that, the architecture model of the corresponding middleware can even change according to the different implementations. For example there are brokered/unbrokered implementations of HLA, or brokered/multi-brokered implementations of JMS. Regarding object model we can see that different middleware systems adopt their own specific object models. For example, within the context of DDS, the pub-sub application is defined as a *domain* which has several *domain participants* that define *Publisher* and *Subscribers* for different *Topics*.

Problem statement

An important issue that usually directly affects the performance of publish-subscribe systems is the allocation of the participants to the available nodes. This is a generic problem that recurs in each of the publish-subscribe middleware systems that we have discussed in the previous section. For small to mid-sized applications with a limited number of participants and several nodes the allocation of the participants can be defined by a human expert. For this, the expert will predict an optimum deployment based on earlier experiences with publish-subscribe systems. But currently software systems are not small scale but easily require a large number of participants and/or nodes. As such, it becomes not tractable anymore for the human engineer to identify an optimal deployment manually.

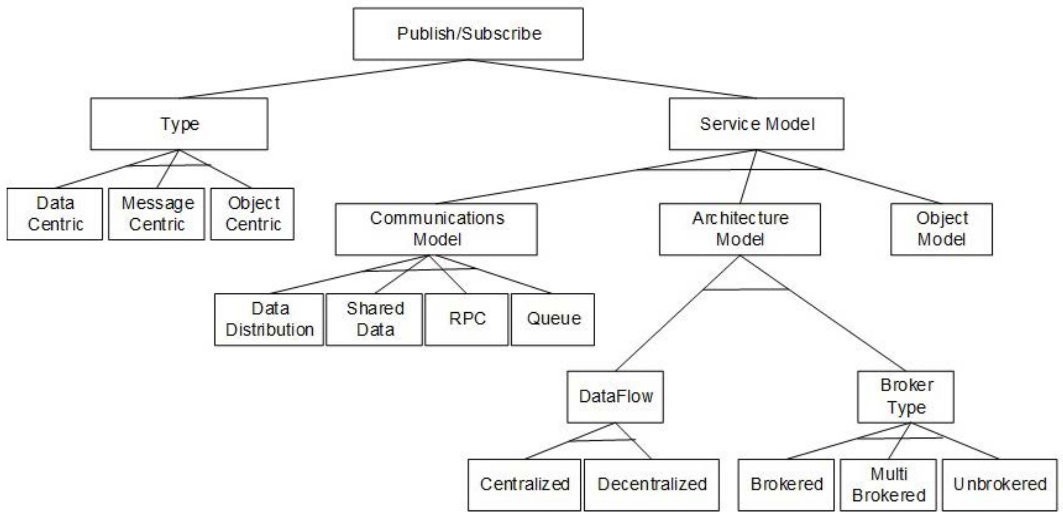


Figure 2: Feature Model of Publish-Subscribe Systems.

Pub/Sub Techn.	Type	Service Model		
		Communication Model	Architecture Model	Object Model
DDS	Data Centric	Data-Distribution	Decentralized/ Unbrokered	Publisher, Subscriber, Domain, Topic, etc.
HLA	Data Centric	Data-Distribution	Decentralized/ Unbrokered or Bro-kered	Federation, Federate, Object Class, etc.
DIS	Data Centric	Data-Distribution	Decentralized/ Unbrokered	Protocol Data Unit (PDU), etc
TENA	Data Centric	Data-Distribution	Decentralized/ Unbrokered	Logical Range, Logical Range Object Model(LROM), etc.
JMS	Message Centric	Queue Based	Centralized/Brokered or Centralized/Multi-Brokered	Queues, Messages, Topics
CORBA Event Services	Object Cen-tric	Remote Procedure Call	Centralized/Multi-Brokered	IDL Objects, Event Channels, etc.

Table 1: Publish-Subscribe Middleware Approaches.

We illustrate this problem in Figure 3, which shows the computed number of possible alternative deployment models according to different node and participant sizes. For three different node sizes (6, 8, 10) we have computed the possible number of alternative deployments. The three functions are also represented in Figure 3. In the figure we can observe that the number of alternative deployments increases exponentially according to participant and node count.

Problem statement

An important issue that usually directly affects the performance of publish-subscribe systems is the allocation of the participants to the available nodes. This is a generic problem that recurs in each of the publish-subscribe middleware systems that we have discussed in the previous section. For small to mid-sized applications with a limited number of participants and several nodes the allocation of the participants can be defined by a human expert. For this, the expert will predict an optimum deployment based on earlier experiences with publish-subscribe systems. But currently software systems are not small scale but easily require a large number of participants and/ or nodes. As such, it becomes not tractable anymore for the human engineer to identify an optimal deployment manually.

We illustrate this problem in Figure 3, which shows the computed number of possible alternative deployment models according to different node and participant sizes. For 3 different node sizes (6, 8, 10) we have computed the possible number of alternative deployments. The three functions are also represented in Figure 3. In the figure we can observe that the number of alternative deployments increases exponentially according to participant and node count.

From a general perspective, finding the feasible deployment alternative is a combi-natorial optimization problem in the branch of optimization or operations research in mathematics. This problem aims to find an optimal alternative from a design space consisting of finite set of alternatives. It is known that in such combinatorial optimization problems, *brute-force search* or *exhaustive search*, in which possible candidates for the solution are systematically enumerated and checked, is not feasible for large design spaces. One way to speed up this brute-force strategy is to reduce the design space

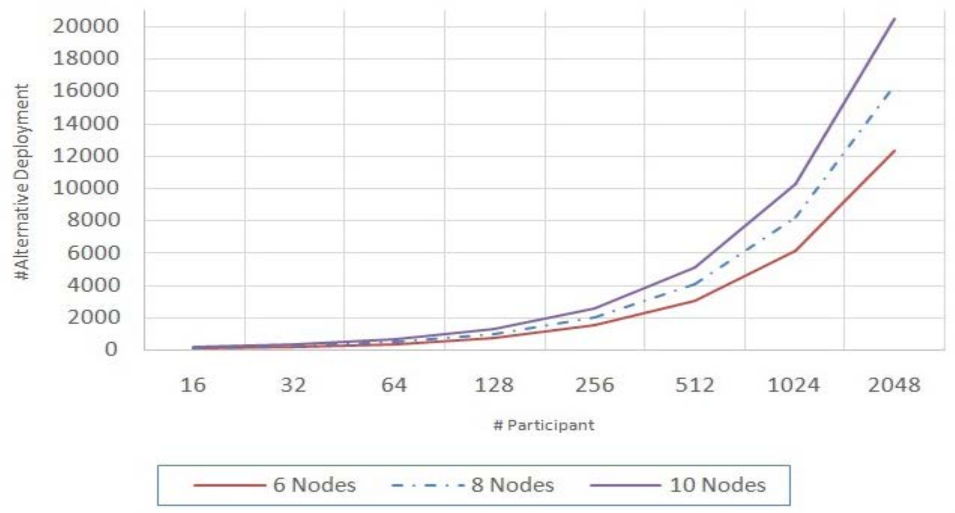


Figure 3: Functions for showing the possible numer of Participant to Node Allocations for node sizes 6, 8 and 10.

of the set of candidate solutions, by using heuristics specific to the problem class. Within the context of finding deployment alternatives of publish-subscribe systems the heuristics can be defined by the human expert based on earlier experiences, but as we have shown in Figure 3 the design space gets simply too complex to be tractable. Hence, other approaches are needed to select the feasible design alternatives. For this, we can first translate the feasible deployment alternative selection problem to the category of so-called *task allocation problem* (TAP)[] [27,19,21], which is one of the fundamental combinatorial optimization problems. In its most general form, the TAP defines as input a number of agents (nodes) and a number of tasks (participants). Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. The objective of the problem is to find a feasible mapping (solution) for the given input. Several different algorithmic solutions have been devised to solve the task allocation problem. In general each TAP algorithm takes as input the required optimization parameters and produces the feasible allocation of tasks to processors. The optimization parameters may include parameters such as execution cost, communication cost, memory consumption and I/O cost.

In our particular case, besides of the definition of nodes and participants, also constraints are imposed on the physical resource properties and likewise the problem needs to be further specialized. Concretely, the feasible deployment problem that we want to solve appears to be an instance of the so-called *Capacitated Task Allocation Problem* (CTAP) [9] which specializes the TAP by including constraints on memory capacity and processing power [15]. Formally, the objective function of CTAP is shown in Figure 4. The problem shown in Figure 4 can be defined as follows:

There exists *m* tasks, where task *i* requires m_i units of memory. There are *n* non-identical processors, where processor *p* has a memory capacity of M_p and processing power of C_p. The cost of executing task *i* on processor *p* is x_{ip}. In addition, c_{ij} denotes the communication cost of tasks *i* and *j*. Communication frequencies shall be taken into account while calculating communication costs. A higher communication frequency between tasks *i* and *j* results in a higher communication cost, c_{ij}. We aim to assign each task to a processor without violating the memory and the processing power constraints of each processor.

The objective in our problem is to minimize the sum of total execution cost and total communication cost (among participants) while not exceeding the memory capacity of each node.

In this paper we do not aim to provide an algorithmic solution by designing novel algorithms or analyzing existing ones. Instead we focus on tackling the problem from an engineering perspective by integrating architectural modeling, algorithmic design preparation and analysis, and model-driven development to generate the required deployment alternatives. Hence our research objective is as follows:

"*Provide an approach and tool support for defining Publish-Subscribe system architecture, extracting the necessary task allocation input parameters for task assignment algorithm from the design using a task assignment algorithm to find an optimized task-to-processor allocation, generating alternative deployment models from the output of task assignment algorithm, and evaluating the generated deployment models*"

Assign tasks to processors to minimize the sum:

$$\sum_{i=1}^{m}\sum_{p=1}^{n} a_{p} x_{p} + \sum_{(i,j)\in E}\sum_{p=1}^{n} a_{p}(1-a_{p})c_{j}$$

Subject to:

$$\sum_{p=1}^{n} a_{p} = 1, i \in T$$

$$\sum_{i} m_i a_{p} \leq M_p$$

$$a_{p} = \{0,1\} \quad p \in P, i \in T$$

(aip = 1, if task i is assigned to processor p, 0 otherwise)

Where:
T, set of m participants = {t1, t2, ...,tm}
P, set of n nodes {p1, p2, ..., pn}
Mp, memory capacity of node p
mi, amount of memory needed for participant i
Xiq, cost of executing ti participant on pq node.
E, set of communication between participants, whereby each communicating participant combination (i, j) has a communication cost cij if participants ti and tj are assigned to different nodes. Communication cost is negligible if two participants are assigned to same node.

Figure 4: Objective function for deriving feasible deployment models.

Case Study

In this section we shortly describe the case study that we will use as a running ex-ample throughout the paper. The case study we have defined is a simulated city with agents like creatures (e.g. people, pets), buildings (e.g. houses, schools), vehicles (car, truck, motorcycle, bicycles, etc.), traffic lights, etc. All agents will be autonomous and will interact with other agents. The decomposition view of the case study is given in Figure 5.

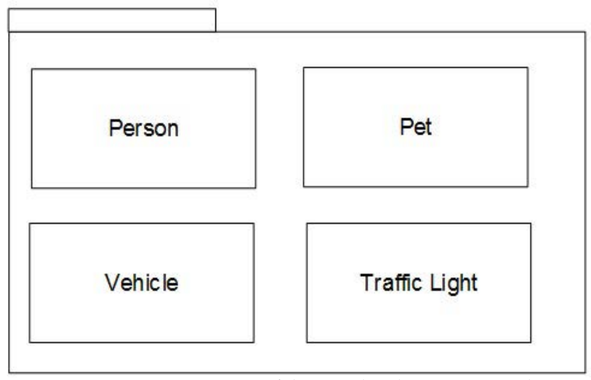


Figure 5: Decomposition View of the Simulated City Case Study.

In a sample simulated city scenario, there may be for example, 600 people, 80 pets, 680 vehicles, and 30 traffic lights. Although the city population is small in this case study, totally there are 1390 participants. We have calculated number of alternative deployments for this sample scenario and results are given in Table 2. It can be seen that even for a small scale scenario, it is not tractable to derive and evaluate alternative deployments manually. In the following section we first describe our proposed method to derive feasible alternatives and use this case study to explain the method steps.

	# of Alternative Deployments		
Participant #	6 Nodes	8 Nodes	10 Nodes
1390	8334	11112	13890

Table 2: Number of Alternative Deployments for the Casestudy According to Node Count.

Method

In the previous section we have discussed the allocation problem that is generic to the existing publish-subscribe middleware systems. In this section we provide a gener-ic method for deriving and evaluating feasible deployment alternatives for these publish-subscribe middleware systems. The method integrates architectural modeling with the algorithmic solution to the CTAP problem to select a feasible deployment alternative, and generate the deployment using model-driven development techniques. The method will be used in the design phase where the system is not developed yet, and the code is not available. The method consists of two basic activities *Architecture Design* and *Feasible Deployment Generation*. We explain each activity below.

Architecture design

The architecture design activity is shown in Figure 6. It starts with the step Define Re-quirements which will provide a description of the required scenarios of the Publish-Subscribe system. This is the only step that is manual, for the remaining steps of the method we provide tool support as discussed in the next section.

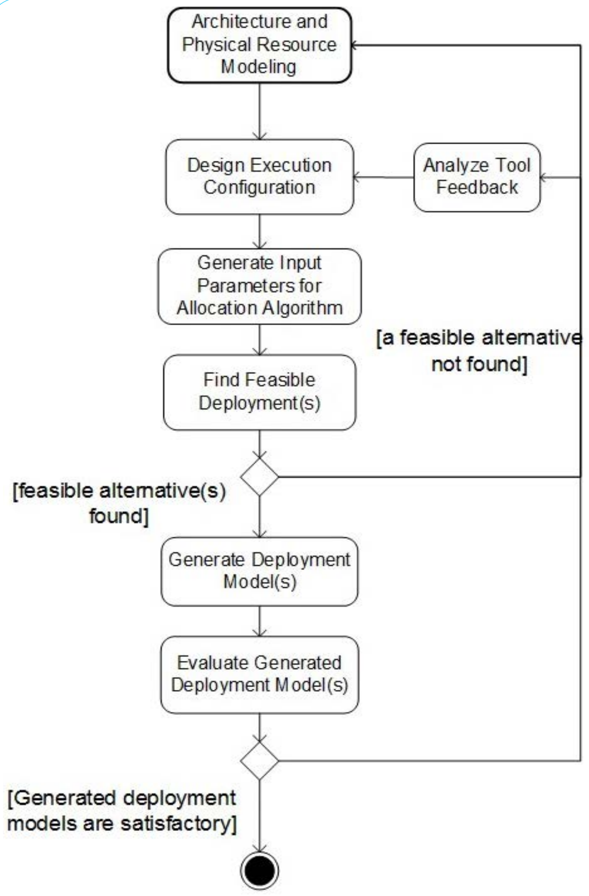

Architecture and Physical Resource Modeling
Design Execution Configuration
Analyze Tool Feedback
Generate Input Parameters for Allocation Algorithm
[a feasible alternative not found]
Find Feasible Deployment(s)
[feasible alternative(s) found]
Generate Deployment Model(s)
Evaluate Generated Deployment Model(s)
[Generated deployment models are satisfactory]

Figure 6: Feasible Deployment Generation Steps.

In the step *Define Data Exchange Model*, the data exchange model is defined to support the publish-subscribe communication and ensure type-safe data exchange among participants. The step *Define Participants* defines the required participants based on the requirements. *Define Pub/Sub Relations* defines the publish-subscribe relations of the participants based on the defined data exchange model. Parallel to these steps the nodes and their network connections are designed together with the property values for processing power and memory capacity. Once the architecture is designed the *Feasible Deployment Generation* activity can be started. For the steps Define *Data Exchange Model, Define Participants*, and *Define Pub/Sub Relations* tool support is provided to assist the user in defining the corresponding models. The step Feasible Deployment Generation is fully automated after the required input is provided.

Feasible deployment generation

Figure 6 shows the steps for defining the steps for deriving and generating the feasible deployment model of the system. The step *Design Execution Configuration* defines the run-time properties of the pub/sub application defined in the architecture design phase. This includes the definition of the number of participant instances, the definition of the update rate for participant instances for each publication (in the publish/subscribe definition), and the definition of the execution cost of each participant instance on each target node. The step *Generate Input Parameters for Allocation Algorithm* defines the required input parameters values for defining the possible allocations in the implemented allocation algorithm. For this, both the static and run-time properties of the participants and the physical resources are defined. The algorithm for computing the feasible deployment alternatives is executed in step *Find Feasible Deployment Model(s)*. If the algorithm can find a feasible deployment this is provided as a table representing the mapping of tasks (participant 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. There is a trade-off between resource consumption and communication costs. In case more modules are deployed on a single node, while the resource consumption will increase, the communication costs will decrease and vice versa. If no feasible solution was found in the previous step, detailed feedback is presented to the designer to optimize the design model in the step *Analyze Tool Feedback*. Based on a trade-off analysis the designer will then first try to update the 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.

1) A User selects the files for uploading/downloading with <input type = ”file”> element in the HTML <form> element.

2) The files are transmitted with the POST method.

3) The JSP file describes the destination.

4) The JSP file receives the request and analyzes the original file name for each file.

5) The JSP file transforms the file into an object of ”Input-Stream” to the Java binary code.

Metamodel for Publish-Subscribe Systems

In this section we provide the metamodel for Publish-Subscribe systems that will be used for supporting the method as well as the tool set Deploy-PS that we describe in the next section. The developed metamodel as shown in Figure 7 is generic and represents the basic concepts for Publish-Subscribe systems. For representing special Publish-Subscribe such as DDS and HLA it needs to be specialized. The metamodel is built around four basic elements that we will discuss in the following subsections.

Application model

This model element defines the element for defining the Publish-Subscribe applica-tion. The application model consists of a set of *Participants, DataExchangeModelElements*, and *PubSubRelations* to define the Publish-Subscribe system participants, elements of the data exchange model and Publish-Subscribe relations among participants and data exchange model elements. The possible Publish-Subscribe relation types are defined in *PubSubTypeEnum* as *Publish, Subscribe*, and *PublishSubscribe* (means both publish and subscribe). This metamodel element is instantiated in the step *Define Pub-Sub Application* of the method as shown in Figure 8.

Physical resource model

This model element contains metamodel elements that will be used for defining the node properties and is instantiated in the step *Design Node Properties* of Figure 8. Physical resource model contains a set of Nodes to represent each processing element. Each Node defines *memoryCapacity* property and has a set of *Processors* to represent the values for the capacity and computation power. In addition to the node properties, the physical resource model also defines the network connections among the nodes with *Network* and *NetworkConnection* elements.

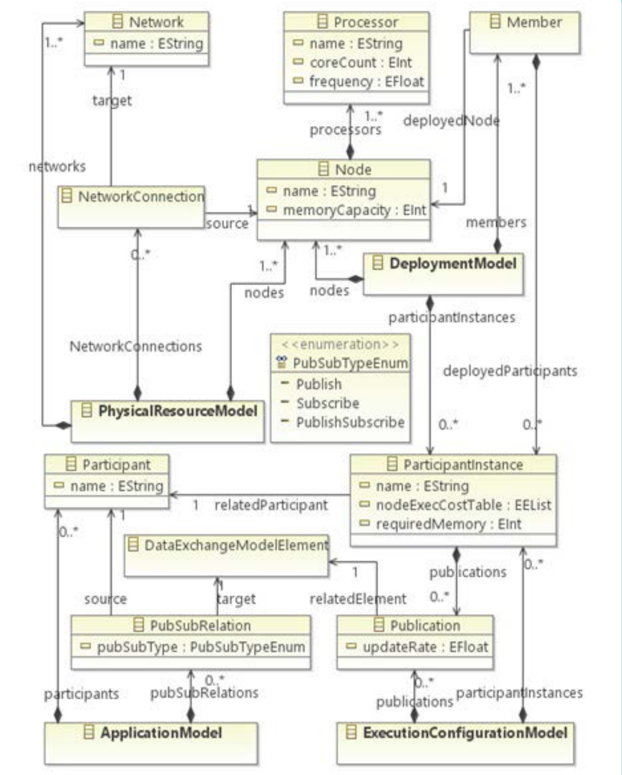


Figure 7: Abstract Metamodel for Publish-Subscribe Systems Feasible Deployment.

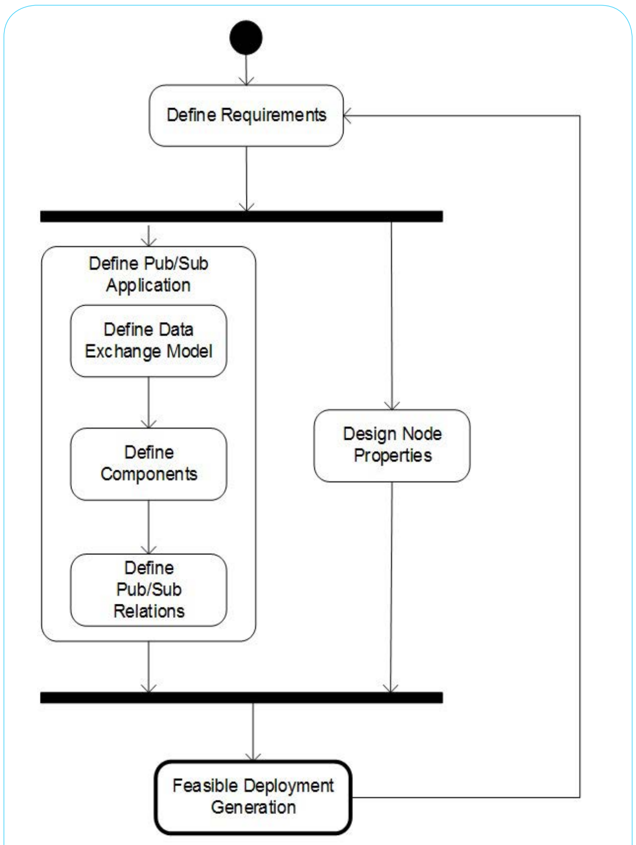

Define Requirements
Define Pub/Sub Application
Define Data Exchange Model
Define Components
Define Pub/Sub Relations
Design Node Properties
Feasible Deployment Generation

Figure 8: Architecture Design Activity of the Method.

Execution configuration model

This model element contains the metamodel elements that will be used for defining the execution configuration models given in Fig. 7. An execution configuration defines the dynamic properties of a Publish-Subscribe system with a number of *Participant Instances* and *Publications*. Each *Participant Instance* relates to a participant in *Application Model*. A *Participant Instance* also defines the required Memory of the participant instance and a *node Exec Cost Table* that represents execution cost of the participant instance on a specified node. Since execution cost of a participant instance may vary according to the node properties, an execution cost table is used instead of defining a constant execution cost value. Publication element defines a participant's publication of a *Data Exchange Model Element* defined in *Application Model* with a specified upda-teRate.

Deployment model

This model element contains metamodel elements that will be used for defining generated deployment models given in Figure 6. A deployment model consists of a set of Members that contains a set of deployed *Participant Instances* (defined in *Execution Configuration Model*). Each Member is deployed on a Node defined in *Physical Resource Model.*

Relation to platform specific metamodels

The metamodel that we have defined in Figure 7 can be specialized to define platform specific middleware systems.

The mapping of the generic metamodel elements to two example middleware sys-tems is shown in Table 3. Some metamodel elements remain the same for the differ-ent platform specific models. For example the metamodel element *Publication* is the same for all the middleware systems as we have shown in Table 1. Some metamodel elements have different names but directly map to the generic metamodel elements. For example, the generic metamodel element *Participant* maps to the metamodel element Federate of HLA, and *Domain Participant* of DDS. Some metamodel ele-ments map to more than one metamodel elements of specific platforms. For example, the *Data Exchange Model Element* maps to Object Class, Interaction Class, etc of HLA.

Generic Element	HLA Equivalent	DDS Equivalent
Pub-Sub Application	Federation	Domain
Participant	Federate	Domain Participant
Data Exchange Model Element	Object Class, Interaction Class, Basic Datatype, Simple Datatype, Fixed Record, Enumeration, Variant Datatype,	Topic, Topic Type, Struct, Sequence, Typedef, etc.
Participant Instance	Federate Instance	Domain Participant Instance
Publication	Publication	Publication
Pub-Sub Relation	Pub-Sub Relation	Pub-Sub Relation
PubSubTypeEnum	PubSubTypeEnum	PubSubTypeEnum
Member	Member	Member
Node	Node	Node
Processor	Processor	Processor
Network	Network	Network
Network Connection	Network Connection	Network Connection

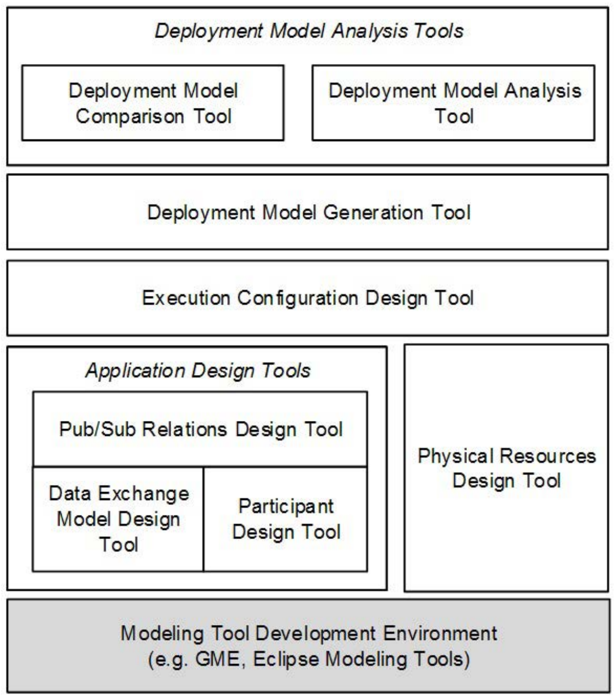
Table 3: Mapping Abstract Metamodel for Publish-Subscribe to Sample Specialized Middleware Systems.

Deploy-PS Tool

Based on the approach that we have defined in section 4 we have developed the corresponding toolset Deploy-PS, which is an integrated development environment for supporting the modeling, generation and analysis of publish-subscribe architectures. The overall architecture of Deploy-S is shown in Figure 9.

Deploy-PS is built on the Eclipse Modeling Tools and is implemented as a set of plug-ins. Eclipse Modeling Tools consists of different tools such as Eclipse Modeling Framework – EMF [16] (modeling framework and code generation facility), Graph-ical Modeling Framework – GMF [17] (graphical editor development framework), Emfatic [18] (a text editor and a language for editing EMF models), EuGENia [19] (an abstraction and model generation tool for easing development of GMF editors).

Deploy-PS consists of eight tools that realize the steps of the approach in section 4, and which uses the metamodel of Figure 7.


Deployment Model Analysis Tools
Deployment Model Comparison Tool
Deployment Model Analysis Tool
Deployment Model Generation Tool
Execution Configuration Design Tool
Application Design Tools
Pub/Sub Relations Design Tool
Data Exchange Model Design Tool
Participant Design Tool
Physical Resources Design Tool
Modeling Tool Development Environment (e.g. GME, Eclipse Modeling Tools)
Figure 9: Overall Architecture of the Deploy-PS tool

Application Design Tools is defined for designing the application model. This tool group consists of three different tools for Data Exchange Model Design, Participant Design, and Pub-Sub Relations design. The *Physical Resources Design Tool* is defined for designing the physical resources. The *Execution Configuration Design Tool* supports the development of execution configurations based on the define application model and physical resources. The *Deployment Model Generation Tool* is used to automatically generate the deployment models based on the application model, physical resources and the execution configuration. The Deployment Model Analysis Tool and Deployment Model Comparison Tool are used for evaluating deployment models with respect to different quality factors.

Evaluation

We have used the case study of section 4 to evaluate our approach and the PS-Deploy tool. We defined the requirements of the case study, defined the necessary models for the case study in PS-Deploy tool environment, generated the deployment models and evaluated the performance of the generated deployment models.

To support the automated selection of the feasible deployment alternative we have defined the city simulation using the Deploy-PS tool. First we defined the data exchange model as given in Figure 10, which includes vehicles, creatures, buildings and other related datatypes such as creature gender, vehicle type, position, etc.

After defining the data exchange model, we have designed the participants and their pub-sub relations with data exchange model elements as shown in Figure 11. Participants publish, subscribe or both publish-subscribe data exchange model elements as defined in Figure 10.

With the definition of the data exchange model, the participants and the pub-sub relations, the application definition phase is completed. In the next step we defined the physical resource model as shown in Figure 12. The physical resource model consists of four connected nodes with different memory capacities and processing powers.

After defining the application model and the physical resources model, we defined the execution configuration model given in Figure 13. The execution configuration de-fines instance count for each participant per the scenario given in requirements and update rates for each publication of participants. The execution configuration model also defines memory requirements of each participant instance and execution costs per each node defined in Figure 12.

After designing the pub-sub application model, the physical resources model, and the execution configuration the deployment model is automatically generated by the Deploy-PS tool. The result is shown in Figure 14. As we can observe from the figure, the PS-Deploy

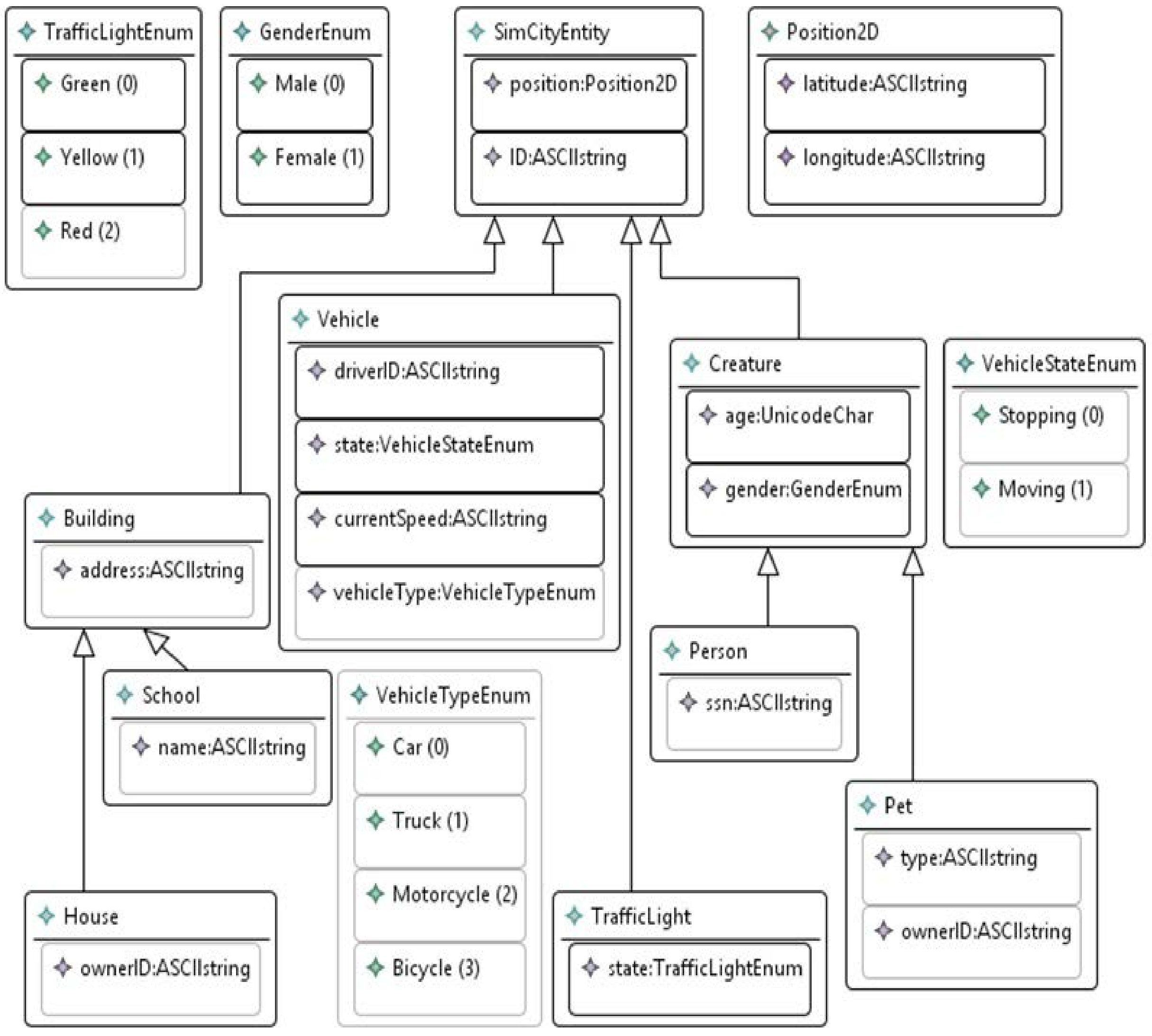


Figure 10: Data Exchange Model for the City Simulation Case Study.

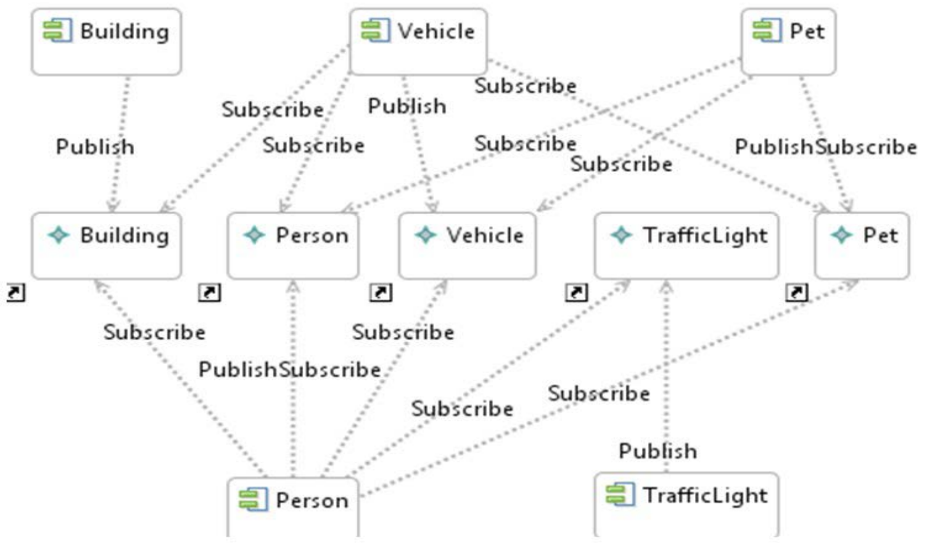


Figure 11: Participant Definition and Pub-Sub Relations with Data Exchange Model Elements for the City Simulation Case Study.

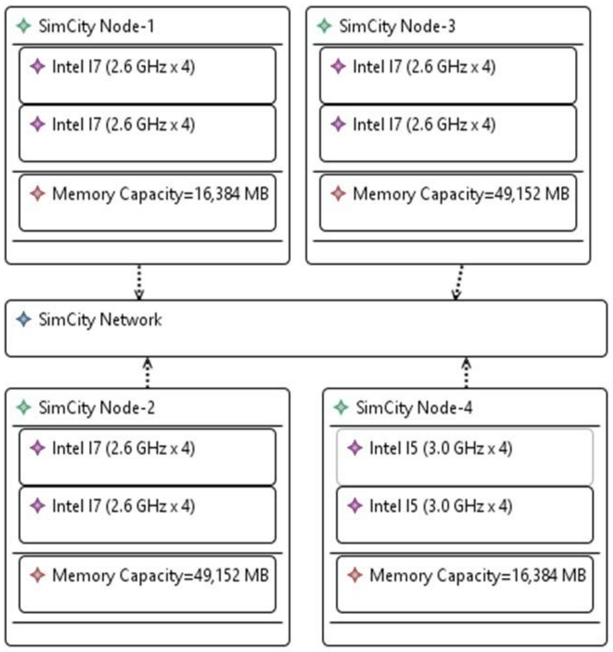


Figure 12: A Sample Physical Resource Model for the Case Study.

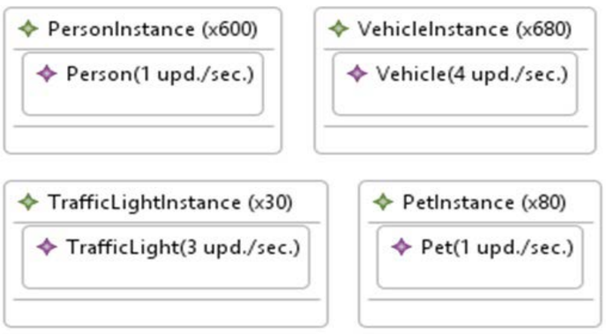


Figure 13: Execution Configuration Model for the City Simulation Case Study.

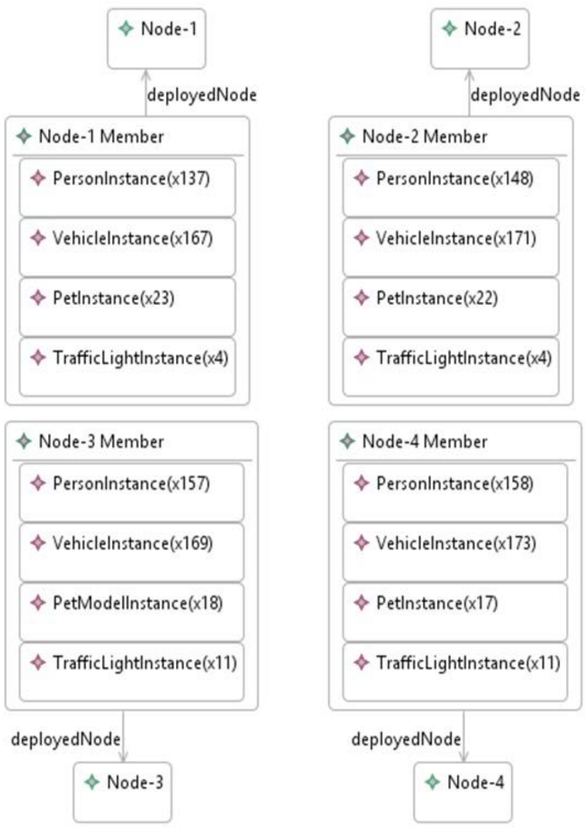


Figure 14: Generated Deployment Model for the Case Study.

tool has allocated participant instances defined in the execution configuration given in Figure 13 to the nodes defined in Figure 12. Some of the vehicles are assumed to be autonomous, so there are more vehicles than people.

We have compared the generated deployment model with a manually defined expert judgment deployment model by using model evaluation tools provided by PS-Deploy tool. The automatically generated deployment model is 15% better than the manually defined model with respect to the total execution cost of the participants and about 25% better in means of total memory requirement. The statistics are in alignment with the measurements that we have conducted in our previous work [5].

In the above paragraphs we have shown the application of the method for simulation systems. However, as we have stated before the method and the tool can also be applied to other Publish-Subscribe Systems. For example, for DDS-based systems [2] we specialized our approach to meet the specific characteristics of the DDS middleware. Different from HLA (where we have developed the metamodels from stratch), we realized OMG's UML profile for DDS for application design tools (see Figure 9). This is an example case that shows modular and generic nature of our approach that enables re-using existing metamodels/UML profiles for different activity steps. More detail about specialization of our approach for DDS can be found in [10].

Related Work

Different architectural evaluation approaches have been introduced to evaluate the stakeholders' concerns. A comprehensive overview and comparison of architecture analysis methods have been given by, for example, Dobrica et al. [20], and Babar et al [21]. Kazman et al. [22] have provided a set of criteria for comparing the foundations underlying different methods, the effectiveness and usability of methods. To compare the architectural evaluation approaches several frameworks have been proposed. The Software Architecture Review and Assessment (SARA) report provides a conceptual framework for conducting architectural reviews. The frameworks compare the methods usually based on the criteria of context and goals of the method, required content for applying the method, the process adopted in the method, and the validation of the method. In this paper we have provided a dedicated architecture analysis approach for analyzing the deployment of publish-subscribe architectures.

Related to our work there are several other approaches in other domains for opti-mizing 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. [23] 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. [23] 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. [24]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. [24] used Mixed Integer Linear Programming (MILP) techniques and used CPLEX [1] as MILP solver.

Aleti et al. [25] 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. Aleti et al. [25] adapt Pareto-Ant Colony Optimization (P-ACO) algorithm [26] 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 [27]. 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. [28] 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. [28] 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.

Švogor, and Carlson, use heuristics and Analytic Hierarchy Process (AHP) [29] for weighted multi-objective design space exploration [30]. The main objective of this study is to support systems architects in complex allocation decisions in the early design phases.

Bahrami-Bidoni et al. propose an algorithm to find the best possible allocation of parallel application tasks to processors in heterogeneous distributed environment [30]. Hereby, the authors focus primarily on minimizing the computation time and provide an evaluation ofthe performance of the proposed algorithm for different problem types including task interaction density, problem size, and communication to computation time ratio.

In the context of task allocation in heterogeneous distributed environments, Kang et al. propose an approach with the aim to maximize system reliability [31]. The pro-posed approach is based on the greedy search algorithm. Like other approaches, the performance of the proposed solution depends on the characteristics of the problem such as the number of tasks & processors, task interaction density of application, and average communication to average computation time ratio.

Dad et al. focus on multi-simulation graph distribution on multi-core clusters for a FMI (Functional Mock-up Interface) [32] compliant multi-simulation environment for continuous time systems [33]. The authors performed experiments on two clusters, running up to 81 simulation components (FMU) and using up to 16 multi-core computing nodes to measure effectiveness of graph distribution.

Conclusion

Architecting large scale publish-subscribe systems is not tractable due to the large design alternative space and the trade-offs between different parameters such as execution cost and communication cost.

In this paper we have provided a general and systematic approach together with the corresponding toolset *Deploy-PS* for finding the feasible deployment of participants to nodes in publish-subscribe systems. With the current approach we have achieved to solve the deployment problem for a broader set of applications based on publish-subscribe systems. The automatic generation of the various architecture design alternatives supports the architecture trade-off analysis for deriving a feasible solution. Our quantitative evaluations have shown that both the approach and the *Deploy-PS* are very useful in designing and evaluating publish-subscribe architectures. From an algorithmic perspective the problem that we have addressed is known as the Capacitated Task Allocation Problem. However, to provide an effective solution we had to provide lots of effort in aligning the algorithm to the architectural model and application, defining and implementing the required metamodels, implementing the required model-transformations and realizing the overall toolset. In our future work we will aim to integrate various different algorithmic implementations of the CTAP, and customize our metamodel and toolset for specific publish-subscribe infrastructure.

Competing Interests

The authors declare that they have no competing interests.

References

1. Eugster P, Felber PA, Guerraoui R, Kermarrec A (2003) The many faces of publish/subscribe. Journal ACM Computing Surveys 35: 114-131.
2. OMG (2007) Data distribution service for real-time systems, Ver 1.2.
3. Oracle (2002) Java message service specification, Ver 1.1.
4. IEEE (1998) IEEE STD 1278.1A-1998: Standard for distributed interactive simulation – application protocols.
5. IEEE (2010) IEEE STD 1516-2010 Standard for modeling and simulation (M&S) High Level Architecture (HLA) - framework and rules.
6. Pirim T (2006) A hybrid metaheuristic algorithm for solving capacitated task allocation problems as modified XQX problems, The University of Mississippi). ProQuest Dissertations and Theses.
7. Lo VM (1988) Heuristic algorithms for task assignment in distributed systems. IEEE Transactions on Computers 37: 1384-1397.
8. Çelik T, Tekinerdogan B (2013) S-IDE: A tool framework for optimizing deployment architecture of high level architecture based simulation systems. Elsevier Journal of Systems and Software 86: 2520- 2541.
9. Çelik T, Tekinerdogan B, Imre K (2013) Deriving feasible deployment alternatives for parallel and distributed simulation systems. ACM Transactions on Modeling and Computer Simulation Journal, vol 23, 3, Article 18 , July 2013.
10. Celik T, Koksal O, Tekinerdogan B (2014) Deploy-DDS: Tool Framework for Supporting Deployment Architecture of Data Distribution Service based Systems, In Proceedings of the 2014 European Conference on Software Architecture Workshops (ECSAW '14). ACM, New York, NY, USA, Article 35.
11. Foster A (2013) Messaging Technologies, A comparison between DDS, AMQP, MQTT, JMS and REST, v1.1, PrismTech Whitepaper .
12. Pardo-Castellote G (2008) Introduction to DDS, OMG Real-Time Workshop.
13. Pardo-Castellote G (2011) DDS:The data-centric future beyond message-based integration, OMG C4I .
14. Mehrabi A, Mehrabi S, Mehrabi AD (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.
115. Stone HS (1977) Multiprocessor scheduling with the aid of network flow algorithms, IEEE Transactions on Software Engineering 3: 85- 93.
116. Budinsky F, Steinberg D, Merks E, Ellersick R, Grose T (2003) Eclipse modeling framework. Addison-Wesley Professional.
117. Voelter M, Kolb B, Eftinge E, Haase A (2006) From front end to code – MDSD in practice.
118. Daly C (2004) Emfatic language reference.
119. Kolovos DS, Paige DF, Polack F (2006) Eclipse development tools for epsilon. n Eclipse Summit Europe, Eclipse Modeling Symposium.
120. Dobrica LF, Niemela E (2002) A survey on software architecture analysis methods", IEEE Transactions on Software Engineering 28: 638-653.
121. Babar MA, Zhu L, Jeffrey R (2004) A framework for classifying and comparing software architecture evaluation methods. In Proceedings of 5th Australian Software Engineering Conference, April, pp. 309-319.
122. Kazman R, Bass L, Klein M, Lattanze T, Northrop L (2005) A basis for analyzing software architecture analysis methods. Software Quality Journal 13:329-355,.
123. 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.
124. Zheng W, Zhu Q, Di Natale M, Vincentelli AS (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, 161-170.
125. Aleti A, Grunske L, Meedeniya I, Moser I (2009) Let the ants deploy your software - An ACO based deployment optimisation strategy. ASE '09 Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, pp 505-509.
126. Doerner K, Gutjahr WJ, Hartl RF, Strauss C, Stummer C (2004) Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. Annals of Op Res 131: 79-99.
127. Aleti A, Bjornander S, Grunske L, Meedeniya I (2009) Acheopterix: An extendable tool for architecture optimisation of aadl models, in MOMPES'09. IEEE Digital Libraries, pp. 61-71.
128. Malek S, Medvidovic N, Mikic-Rakic M (2012) An extensible framework for improving a distributed software system's deployment architecture. IEEE Trans Software Eng 38: 73-100.
129. T.Saaty, What is the analytic hierarchy process? Mathematical Models for Decision Support, 48:109–121, 1988.
130. Švogor I, Carlson J (2015) SCALL: Software Component Allocator for Heterogeneous Embedded Systems. In Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW '15). ACM, New York, NY, USA, Article 66.
131. Kang QM, He H, Wei J (2013) An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. J Syst Softw73: 1106-1115.
132. Blochwitz T, Otter M, Akesson J, Arnold M, Clauss C, et al. (2012) Functional mockup interface 2.0: The standard for tool independent exchange of simulation models." In Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany, no. 076, pp. 173-184. Linköping University Electronic Press.
133. Dad C, Vialle S, Caujolle M, Tavella JP, Ianotto M (2016) Scaling of distributed multi-simulations on multi-core clusters. In Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on, pp. 142-147. IEEE.

This article was originally published in a special issue:

Software Architecture

Handled by Editor(s):

Dr. Mohammad Alshayeb
Information and computer science Department
King Fahd University
Saudi Arabia