An Integrated Tool for Development of Overlay Services

Yuki Sakai
Graduate School of Information Science and Technology
Osaka University, Japan
y-sakai@ist.osaka-u.ac.jp

Akihito Hiromori
Graduate School of Information Science and Technology
Osaka University, Japan
hiromori@ist.osaka-u.ac.jp

Hirozumi Yamaguchi
Graduate School of Information Science and Technology
Osaka University, Japan
h-yamagu@ist.osaka-u.ac.jp

Khaled El-Fakih
School of Engineering
American University of Sharjah, UAE
kelfakih@aus.edu

Teruo Higashino
Graduate School of Information Science and Technology
Osaka University, Japan
higashino@ist.osaka-u.ac.jp

ABSTRACT

We propose an integrated environment for supporting the development of overlay services. Given a service description written in a centralized and network-independent way using a high-level Petri net, our tool automatically derives its distributed version taking into consideration the targeted overlay network specification and network/computing resources. Furthermore, our tool interprets the distributed version and allows the overlay nodes to execute the service as specified in the description. During that time, the tool monitors the utilization of overlay links and occupation of processors so that related information can be provided to the developers. Consequently, the developers only give the service description and our toolset supports the subsequent design and development tasks. An experimental study on using the toolset and a realistic application example are provided to show the effectiveness of our methodology.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Petri nets*

Keywords

Distributed systems, Overlay networks, Petri Nets simulation, Simulation tools

1. INTRODUCTION

High speed broadband networks with high performance computers have brought us a novel computing concept where services are provisioned by cooperative peers connected via virtual connections in a decentralized manner. In such a configuration of service platform, highly-dedicated services can be designed and deployed flexibly by combining several simple services provisioned by different service providers. As an example, let us consider a content sharing service among several online video service providers. We assume that users may require video with appropriate format and quality according to the capability of their terminals (PCs, mobile phones or others) and their access bandwidth. For provisioning a highly functional service, the service may be designed in this way; the requested video is searched over the different online video servers, and if a requested video is found and if its quality or format is not appropriate to the configuration of the requester's terminal, the video is transcoded and the result is returned to the requester.

The development of such overlay services is not an easy task due to the following reasons. First, we need to distribute the service on overlay networks consisting of many servers. However, considering the availability of the network and computing resources, it is not easy to deploy the service in such a way that the performance of the service is maximized. Second, to evaluate the performance of the deployed service, we need to observe the link utility, computation load of nodes and throughput of the service in real distributed environments, which is not easy as well. Furthermore, we need to setup the environment of experiments, implement the distributed codes, locate the codes on the overlay nodes, establish connections among the nodes, and collect logs for the evaluation of the performance.

In this paper, we propose an integrated environment for supporting the development of overlay services. In particular, we use CPN Tools [14] for writing services in a centralized and network-independent way. Then, our tool analyzes a given service description and automatically derives a distributed version of the service taking into consideration the targeted overlay network specification. Our tool interprets the distributed version of the service and let the overlay nodes execute the service as specified in this description. During that time, the tool monitors the utilization of overlay links and occupation of processors so that related information can be provided to the developers. In particular, our tool is equipped with the resource optimizer, which determines the best way to execute the service in the targeting overlay network to maximize the performance of the service. As an application example, we have developed a video transcoding service on an overlay network consisting of four overlay nodes using our environment. The experi-

SIMUTools 2009, Rome, Italy

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

mental results have shown that our environment was very helpful to shorten the response time of the service.

Several frameworks and related toolsets have been proposed and developed for similar or different objectives. For designing service overlay, Refs. [6, 17, 18] have dealt with service design methodologies, and Refs. [1, 11, 12, 15, 16] have developed some support tools for development of overlay networks. Compared with these methodologies or tools, our contribution is summarized as follows. First, we provide a toolset with several unique features in supporting high level design of service overlay. Using the algorithm presented in our previous work [19], the tool can automatically derive a distributed service description from a given centralized service description written in high-level Petri nets. Also the tool has the resource optimizer that allows to optimize the performance of the service. By these features, developers can design the optimized services using powerful GUI from CPN Tools, without knowing the overlay network specification. Second, the tool has several functions for program deployment, debug and performance monitoring. Finally, using a realistic example service, we have conducted experiments in real distributed environment to demonstrate the advantages of our tool. A more detailed discussion on the advantages of our integrated environment along with some related methodologies will be given in Section 6.

2. SERVICES IN OVERLAY NETWORKS; ARCHITECTURE AND EXAMPLE

An *overlay network* consists of *overlay nodes* (or simply *nodes*, hereafter) with network connections between every pair of nodes. Thus, an overlay network can be modeled as a complete graph. A *service* is a computation flow consisting of a set of "*subtasks*". In our methodology, developers can write the service as a centralized program, but when the service is deployed on an overlay network, the overlay nodes need to execute the computation flow in distributed environment. For the distributed execution, we assume that (i) each subtask is executed by one of the overlay nodes, (ii) each data needed to execute the subtasks is located on one of the overlay nodes, and (iii) every overlay node is capable of receiving data needed for executing its own subtasks, executing the subtasks and sending the modified or generated data to some other nodes. Under this assumption, in the distributed execution, it needs to be determined how these overlay nodes should behave to provide the given service.

As an example of a distributed execution of a service on an overlay network, we consider a network of four fully-connected overlay nodes as shown in Fig. 1 and a "Video for Mobile" service (VfM service in short) that provides a video repository service for mobile users as shown in Fig. 2. This service uses two video repositories "A" and "B", which are video databases for PC users and for mobile users, respectively. The service also uses an index database that holds information about the video contents stored in these repositories. Whenever a video is requested with a set of keywords, the service first retrieves the video entries from the index that matches the given keywords and returns the best-matched entry. If the requested video is stored in repository "B", then the service just obtains the content from repository "B" and provides it to the requester. If the requested video is stored only in repository "A", which means that the content is not prepared for mobile users, then the service obtains the content from repository "A", downsizes the video to fit for mobile users, and then provides the downsized video to the requester. In this case the downsized video is registered to repository "B" accordingly. Finally, if no video entries match the requested video, then the service just informs the requester by sending a related notification message.

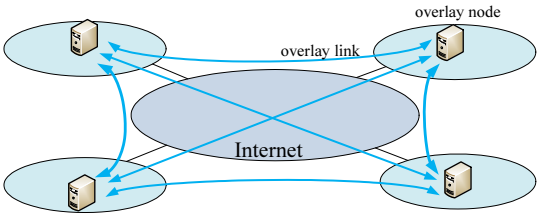

Figure 1: Overlay network architecture.

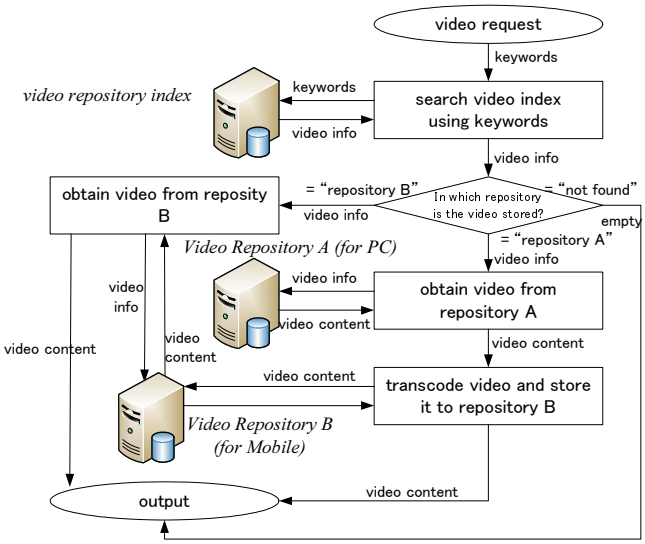

Figure 2: Overlay service example ("Video for Mobile" (VfM) service); several services like video content server, video transcoding and indexing are organized to provide highly-structured services.

In this example, we assume that the index and the repositories are located on different overlay nodes for some technical reasons such as for better load distribution or fault tolerance. Another possibility is that these resources are owned and managed by different service providers in different locations. In such a distributed environment, a possible execution of the VfM service can be as follows: First, an overlay node that acts as the service access point accepts a request with keywords from a requester. The request is then forwarded to the overlay node that contains the index, which in response retrieves entries that match the given keywords. According to the result, one of the overlay nodes that holds repository "A" or "B" retrieves and transcodes the content if necessary. The retrieved/transcoded content is sent back to the overlay node that received the request which in turn notifies appropriately the requester. In this execution, each retrieval or trancoding corresponds to a subtask of the service.

In our previous work [19], we have proposed a method for modeling overlay services that include conditional branches, parallel and/or sequential executions of subtasks involving parameters (resources). In particular, we proposed modeling overlay services using high-level Petri nets, namely the Predicate/Transition nets. We also presented an algorithm to derive distributed specifications of the services. In this paper, based on this work, we provide an in-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

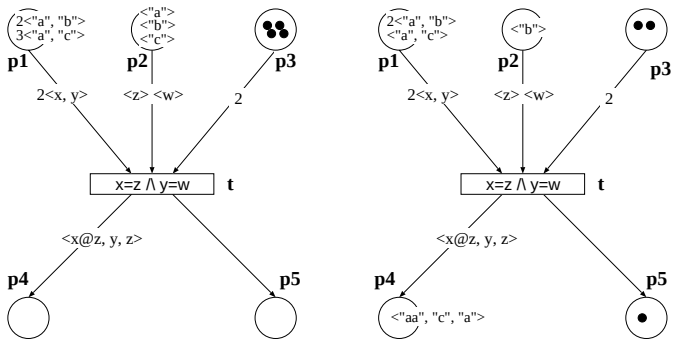

Figure 3: Example of Predicate/Transition net.

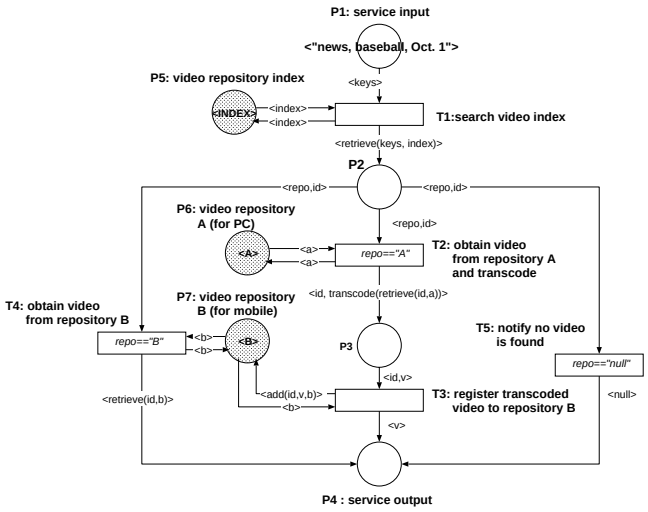

Figure 4: VfM service of Fig. 2 formally written in Pr/T-net.

tegrated tool for supporting the development of overlay services. This tool helps not only in designing distributed overlay services but also optimizing and evaluating the performance of the services in real distributed environment.

3. MODELING OVERLAY SERVICES

In this section we introduce the Predicate/Transition nets (Pr/T-nets) [4, 5] model used for writing overlay services and we briefly introduce the method proposed in our previous work [19] to derive a distributed version of a given service. The tool presented in this paper is based on this method.

3.1 Predicate/Transition Net

In this paper we deal with services written as Pr/T-nets. Pr/T-nets are extended Petri nets where tokens have values and the firability of transitions may depend on those values. We note that Coloured Petri Net (CPN) [8] is a known high-level Petri net that falls into this category. These models have enough modeling power, analytical power and tool support (such as CPN Tools [14]) to specify, verify and analyze large and practical software systems [2], communication protocols [3, 7], control systems and so on [8, 13].

In Petri nets, a place (denoted as a circle) and a transition (denoted as a rectangle) may represent data (or system state) and a task, respectively. A place and a transition may be connected by a directed edge called an arc (denoted by an arrow). Tokens (denoted as black dots) in places represent the current state of the system, and execution ("firing" in the Petri net terminology) of a transition may consume/produce tokens from/to the places connected to the transition.

Formally, in Pr/T-nets, each incoming arc to transition t from place p has a label (called an *arc label*) of the form of $k_1X_1k_2X_2...$ where k_i is a positive integer, X_i is a n-tuple of variables like $\langle x_1, x_2, ...x_n\rangle$ and n is an arbitrary non-negative integer assigned to place p. Place p may have tokens, each of which is a n-tuple of values $C_i = \langle c_1, c_2, ...c_n\rangle$. A set of tokens which can be assigned to the label of an incoming arc to transition t is called an *assignable set* of the arc. Moreover, a transition t may be associated with a logical formula of variables from the labels of incoming arcs of t, called a *condition*. Conditions are depicted inside transitions rectangles. A transition t may fire *iff* there exists an assignable set in each input place of t and the assignment of values to variables by the assignable set satisfies the condition of t. Also, each outgoing arc from transition t to a place p' has a label of the form of $k'_1Y_1k'_2Y_2...$ where k'_i is a positive integer and Y_i is a n'-tuple of values, variables on the incoming arc labels of t or functions over the variables. Therefore, if t fires, the values of the labels on the outgoing arcs from t are determined by the assigned input tokens according to the output arc labels. New sets of tokens are generated and put into the output places of t.

Table 1: Example of place location table for VfM service.

Places	Role	Location
P1	service input	Node 1
P4	service output	
P5	video repository index	Node 2
P2	index search result	
P6	video repository A	Node 3
P3	repository search result	
P7	video repository B	Node 4

In Fig. 3(a), the incoming arc to t from p_1, (p_1, t), has the label $2\langle x, y\rangle$ where x and y are variables. This means that two tokens each consisting of a pair of values are necessary in place p_1 for the firing of transition t. Here, since the following assignable sets $2\langle$"a","c"$\rangle$ in p_1 ("a" and "c" are strings here), $\langle$"a"$\rangle$ and $\langle$"c"$\rangle$ in p_2 and two tokens without values in p_3 satisfy the condition of t, $(x = z \wedge y = w)$, t can fire using these sets. Note that tokens without values are represented as black dots in the following figures. After the firing of t, new tokens are generated to the output places p_4 and p_5 using those token values. The marking after the firing of t is shown in Fig. 3(b). Note that "@" is a concatenation function of two strings. Thus a tuple of strings "aa", "c" and "a" is generated to p_4. The arc label "1", which means the delivery of one token without values, is omitted in the following figures.

3.2 Overlay Service Description

Fig. 4 shows an example Pr/T-net which describes the VfM service of Fig. 2. Places are used to represent service access points, persistent contents (or data repositories), and variables such as those used to hold values of computations. In Fig. 4, places $P1$ and $P4$ are introduced to represent the input and output ports of service requests, respectively, places $P6$ and $P7$ are introduced to represent repositories "A" and "B", respectively, and places $P2$ and $P3$ are introduced to hold the results of retrievals of the index

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

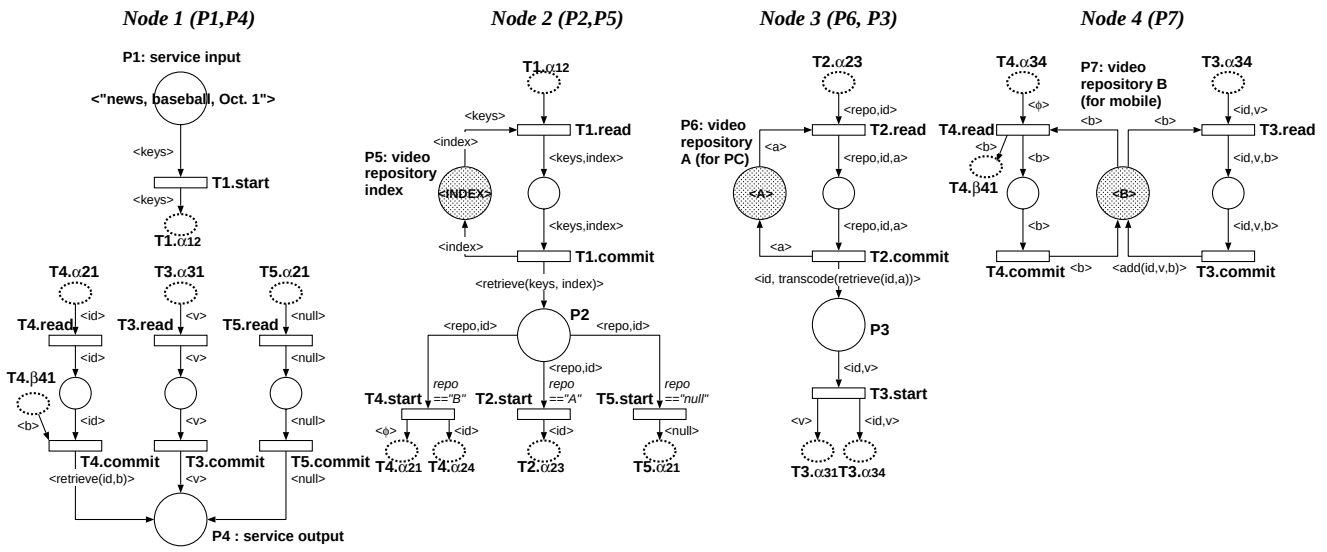

Figure 5: Distributed VfM service of Fig. 2 under place location of Table 1.

and repositories, respectively. In addition, we use the label of an arc from a transition to a place to represent a subtask of the service and also use the transition to represent synchronization of the executions of those subtasks. For example, transition $T2$ has places $P2$ and $P6$ as input places and places $P3$ and $P6$ as output places. The label $\langle id, transcode(retrieve(id, a)\rangle$ of arc $(T2, P3)$ represents the subtask which transcodes the video content (the value of $retrieve(id, a)$) obtained from repository “A” into a downsized one. Transition $T2$ also has another label $\langle a\rangle$ of arc $(T2, P6)$ which represents the subtask that returns the video content taken from repository “A” to the original place. On firing of $T2$, these two subtasks are executed in parallel.

3.3 Place Allocation to Overlay Nodes

One of the important features of our methodology is to decouple service descriptions from overlay network specifications. Accordingly, developers are allowed to write services without being aware of the actual locations of data. That is, they can refer to and update tokens in any place from any transition in writing services. To accomplish this, the method given in [19] provides a way for distributing the given service into overlay nodes knowing an allocation of the places to the overlay nodes. However, it is clear that the performance of a service is affected by this allocation of places. For example, allocating a frequently-accessed database into an overlay node with limited computation power usually leads to a performance bottleneck. Accordingly, in [19], before distributing a given service into overlay nodes, an optimal allocation of places into the overlay nodes is determined using a given optimization model. The model takes into account the computation power of the overlay nodes, the network capability, and the execution orderings of subtasks in given services and simulation experiments were conducted to see the effectiveness of the optimization. The reader may refer to [19] for a detailed description of the optimization model.

The tool presented in this paper implements this optimization model. In addition, optimization is done using collected information about the performance of the overlay nodes and the network. This helps in optimizing the performance of a distributed service as will be demonstrated in Section 5.

3.4 Deriving Distributed Version of Service

We briefly explain the algorithm for deriving a distributed version of a given service. The algorithm inputs a Pr/T-net description of a service and an allocation table of places into overlay nodes (called *place location table*), and outputs a distributed version of the service which is a set of Pr/T-nets that contain communicating behavior between the nodes. Each obtained Pr/T-net corresponds to the behavior description of an overlay node.

We assume that two places with a common name “$X_{u.ij}$” (X is used in the derivation algorithm and is α or γ [19]) in the Pr/T-nets of two different nodes where i and j represent the end points (send and receive buffers) of a reliable communication channel from node i to node j. If a token is put on place “$X_{u.ij}$” at node i, the token is eventually removed and put onto the same place “$X_{u.ij}$” at node j. These places are called *communication places*, and are like “fusion places” in coloured Petri nets [8]. Note that u means that these communication places are used with respect to the execution of transition T_u of the service. In the following figures, communication places are represented as dotted circles with their names inside. Also, in a distributed version of a given service, we introduce a reserved symbol denoted by ϕ, used in tokens for notification purpose only.

Fig. 5 shows a distributed version of the service of Fig. 4 on four overlay nodes based on the place location table in Table 1. The reader may refer to [20] for a complete description of the algorithm. Here, we provide a simple example that demonstrates how nodes can collaborate for providing a given service in a distributed environment and how such cooperative behavior is described using Pr/T-nets with communication places. In the algorithm, one of the overlay nodes that has input places of a transition is selected as the node that starts the execution of that transition (such a node is called the *primary node* of the transition). In Fig. 5, node 1 is selected as the primary node of $T1$ since it has place $P1$. Whenever the input place $P1$ of $T1$ receives a token (a request with keywords), node 1 instantaneously sends this token to node 2 which has place $P5$ through the communication place “$T1.\alpha_{12}$” (it is worth noting that both nodes 1 and 2 include the communication place “$T1.\alpha_{12}$”). Afterwords, at node 2, the retrieval from video index $P5$ is done

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

and the result $retrieve(keys, index)$ is generated as a token into place $P2$ at node 2. This simulates the behavior of $T1$. Place $P2$ represents a choice of $T2$, $T4$ or $T5$ based on the token value of variable '$repo$" in $P2$, and this value has been determined as the result of the previous computation $retrieve(keys, index)$. The reader may verify that the Pr/T-nets of Fig. 5 simulate/realize the service of Fig. 4.

4. INTEGRATED ENVIRONMENT

Our integrated environment mainly consists of the following functional components; (i) *distributed service generator* that derives distributed versions of services, (ii) *resource optimizer* that is the part of the distributed service generator and suggests the "optimal" allocation of places for better performance of the services, (iii) *service executer* that interprets distributed service descriptions written in Pr/T-nets and executes the services on overlay nodes, and (iv) *resource monitor* that monitors overlay nodes and link status.

4.1 Distributed Service Generator and Resource Optimizer

The distributed service generator takes as an input a service written using CPN Tools, coded according to CPN Tools XML format, and a description file of the given service parsed using our dedicated/special XML parser designed for this purpose. Moreover, the generator also takes inputs regarding the overlay network such as the number of overlay nodes and the nodes identifiers, IP addresses and ports.

The service description and the overlay service information are given to the resource optimizer which generates, from the give information, an allocation of places, represented as a place location table, that optimizes the performance of the service. Optimizing the performance is done using the integer linear programing (ILP) model given [19] and using the optimization functions provided later in paper. We use CPLEX to solve the optimization problem and obtain a place location table that achieves the best performance in terms of the considered objective functions. Based on the derived place location table, the corresponding distributed service description is then derived by the distributed service generator. However, due to network uncertainty, the optimized place allocation does not always provide a good performance. To obtain better performance, we provide more information about the overlay networks' status, such as the overlay link status (delay and bandwidth) and the overlay node status (CPU load) which are usually difficult to predict in advance. Thus, we may need to run again the optimization problem and evaluate, by monitoring the execution of the service, the obtained results using the real network . We show how our toolset achieves this goal in Section 5.

4.2 Service Executer

The architecture of the serive executer function is shown in Fig. 6. This function is realized by the cooperation of service access point module, a controller, and a set of overlay nodes. The controller consists of an interpreter of the distributed services, written using CPN tools format, and a system controller that handles the exchange of data among the overlay nodes and the execution of subtasks according to the commands provided by the interpreter. The service access point provides users with Web interfaces for accessing services and also has a system interface module that allows the exchange of input/output data between users and services.

The system interface receives requests from users through the Web-based user interface. Then, it requests the CPN interpreter to put a corresponding token into the service. This request is passed through the system controller. The system interface also requests

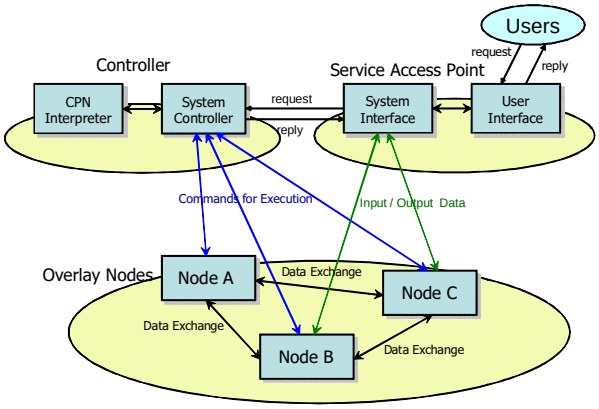

Figure 6: Overview of service executer.

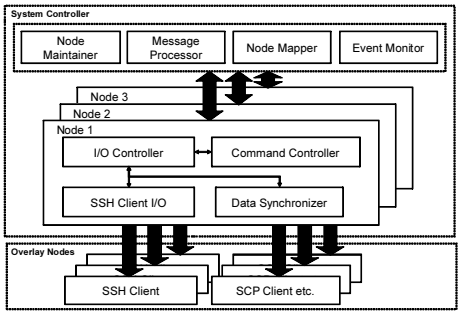

Figure 7: System controller.

the controller to put corresponding data onto the overlay nodes. After executing the service, the system interface receives the output of the service and replies back to the system user. The output is provided to the user as a URL.

The *system controller* is designed as shown in Fig. 7. The controller manages the execution of at the overlay nodes by receiving commands from the CPN interpreter. When the system starts, it initializes the CPN interpreter and registers the distributed service at the interpreter. Also it sends the (compiled) subtask program codes of the service to the overlay nodes that execute these subtasks. The interpreter also sends to every overlay node an initialization shell script that makes the node ready to execute the service by establishing overlay links with other nodes. Managing connections between the system controller and the overlay nodes is done by the *Node Maintainer*. Moreover, the association of overlay nodes of a Pr/T-net service with the actual overlay nodes is done by a *Node Mapper*. Every connection is established using the Secure SHell (SSH) program and the inter-process communication port of the system controller is redirected to the standard I/O of the SSH program.

Message Processor is a message handler that communicates the control messages between the system controller and the overlay nodes. Finally, the *Data Synchronizer* collects the log files recorded at every overlay node using the SCP or the RSYNC programs.

The system controller has an *Event Monitor* that intermediates the CPN interpreter and the overlay nodes. When the CPN inter-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

Table 2: System controller's commands for overlay nodes.

command	receiver node's behavior
connect $node$	send a connection request to $node$
accept $node$	wait for and accepts a connection request from $node$
disconnect $node$	disconnect the connection with $node$
suspend	suspend execution
resume	resume execution
transmit $file$ $node$	send $file$ to $node$
execute $prog$	execute $prog$
sync	execute clock sync. module to synchronize the nodes' clocks with each other
exit	quit from service

preter finds an executable transition, the transition is fired and the programs that correspond to its subtasks are executed at the corresponding overlay nodes. Here, we note that in our model the firing of transitions is not instantaneous since the subtasks associated with transitions (represented as labels of output arcs) may require some execution time. For example, in Fig. 5, $T2.commit$ of node 3 has a function "$transcode(retrieve(id, a))$" on its output arc. The function represents a subtask to downsize the video retrieved from repository A and this subtask may take considerable execution time for transcoding. Similarly, the transmission of large data between overlay nodes (represented as communication places) may require a transmission delay. The event monitor observes the execution of the programs and the transmissions of data, and upon completion, the monitor notifies to the CPN interpreter. The CPN interpreter, after the firing of transitions, delays the generation of tokens to the output places and also delays moving tokens between communication places of different nodes until receiving the notifications from the event monitor. We note that we have developed a dedicated/special CPN interpreter although there are several simulators that can deal with high level Petri nets. This is done to take care of aspects discussed above. In our case, the CPN interpreter needs to interact with the system controller, that controls overall execution of services on overlay nodes, and also needs to deal with multiple Petri nets representing the behavior of overlay nodes.

4.3 Overlay Node Control and Network Management

The overlay nodes execute the programs, which correspond to the subtasks of services including those for exchanging data between nodes, according to the commands given by system controller. These commands are listed in Table 2.

We note that since we allow parallelism in the behavior of an overlay node, while transmitting data from an overlay node, another transmission request with the same destination node may occur. We may delay the latter request till the completion of the current transmission. However, in this case, the response time of the latter request may be long, especially when dealing with large-size files such as video files.

To overcome this problem, we have implemented a multiplexer that allows the sharing of an overlay link among multiple transmissions of data with designated occupancy ratios. Here, the occupancy ratio of one transmission is set as the ratio of the transmission's throughput to the overlay link throughput. To realize parallel transmission of data, the system segments the data into fixed size pieces. Fig. 8 shows an example where three data transmissions are transferred, in parallel, from node A to node B with occupancy ratios 0.25, 0.25 and 0.5, respectively.

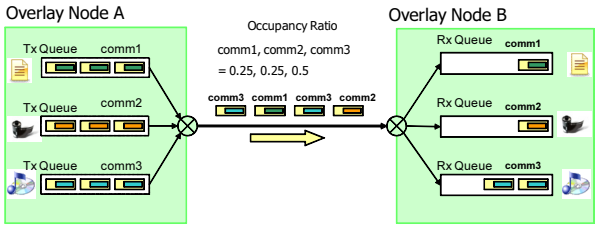

Figure 8: Overlay link control.

Table 3: Performance index of overlay nodes.

Node	Index
Node A	80
Node B	175
Node C	900
Node D	200

5. EXPERIMENTAL STUDY

In order to show the benefits of using our toolset, we provide a software development process that uses our toolset for implementing an example service. We show that our process and toolset facilitate the software development process and contribute to providing an implementation, of a given service, that outperforms the one derived using the existing method given in [19]. In the development process, we first derive and optimize a distributed version of a given service. Then, we deploy the obtained service on an overlay network and evaluate the deployed service and related obtain results. The obtained results are then used for deriving a service with better performance.

As an application example, we consider a movie transformation service where a raw movie is concurrently coded into both a high-quality MPEG2 movie for high-quality playback purpose and a middle-quality MPEG4 movie for network streaming purpose (Fig.9). Also, we assume four computers in the same network segment of 100Mbps LAN. Since the computers have different computing powers, we have measured the execution time of a reference task at each computer and derived their corresponding performance indexes based on the ratios of the inverse of the execution times (therefore larger values indicate higher performance). Table 3 shows these indexes.

In the following, we summarized the development process that we use for deriving a distributed implementation. The process has three phases. In the initial phase, we derive an initial optimized distributed implementation without having any information about the performance of the implemented service and the considered overlay network resources. The initial derived implementation is optimized with respect to the number of messages exchanged among nodes. In the second phase, we first execute the initial derived implementation on the overlay network and record the execution logs of subtasks and then optimize the total execution time of the subtasks knowing the performance indexes of overlay nodes and the executed implementation. However, we observe that after the second phase there is still a possibility to obtain an implementation with a better performance by considering the transmission delay of large-size data. Consequently, in the third phase, we optimize considering the sum of the total execution and transmission times.

5.1 Initial Phase

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

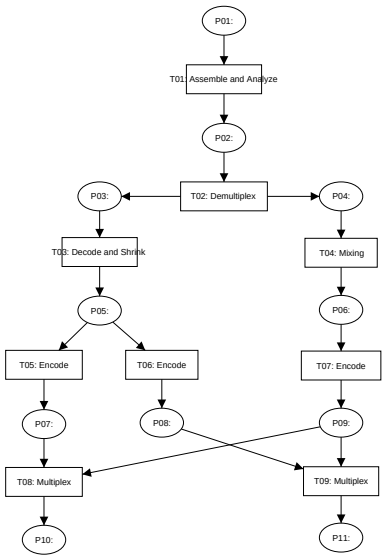

Figure 9: Movie transformation service (we omit transition labels in this figure for simplicity).

Table 4: Place location table (in the initial phase)

Node	Place Assignment
Node A	P01, P02, P03
Node B	P04, P06
Node C	P08, P09, P11
Node D	P05, P07, P10

A given service may handle a large amount of multimedia data. Therefore, in order to execute the service efficiently, we try to make the traffic load between overlay nodes as small as possible. However, at the initial development phase, we do not usually know the overlay network status, *e.g.*, how much throughput can be achieved between nodes. Therefore, we derive an optimal place allocation so that the total number of the data exchanged among overlay nodes can be minimized. For this purpose, we use the ILP constraints given in [19] and the following objective function.

$$F_{min} = \sum_{i,j \in N \times N (i \neq j)} \sum_{t \in T} \{\alpha_{ij}^t + \gamma_{ij}^t\}$$

where N and T denote the set of overlay nodes and the set of transitions, respectively, and X_{ij}^t (X is either α or γ) is a 0-1 variable that represents whether an α (or γ) message from node i to node j for simulating transition t is sent or not. Thus, we derive an optimal allocation of places by minimizing function F_{min} subject to the constraints given in [19]. For the considered service, the corresponding obtained place allocation table is given in Table 4.

Afterwards, in order to evaluate the obtained service under the obtained place locations, we execute the service on the overlay network using our toolset. During the execution of the service, the overlay nodes record the average CPU load to process each subtask and the average network throughput to send data to other nodes.

After the execution of the service, we analyze the execution logs collected from the overlay nodes and provided to the system controller. Fig. 10 shows the timing chart and the network/node status. The upper part of the figure shows when and by which node the subtasks are executed and the corresponding exchanged date, and the lower part shows the CPU load and network throughput of each node at the corresponding time.

Table 5: Place location table (in the second phase)

Node	Place Assignment
Node A	P01, P02
Node B	P03, P04, P05
Node C	P06, P08, P09, P11
Node D	P07, P10

In this experiment, the response time of the service which is the time to return the result to the requester, was 224 seconds (see the x-axis of Fig. 10). The response time of the service is usually determined by the longest path in the timing chart from starting the execution of the service to ending the execution of the service. Such a longest path is called a *critical path* and is a sequence of subtasks (execution and data transmission). In Fig. 10, we observe that the execution time of transition $T03$ is relatively large and $T03$ affects the execution time of its succeeding transitions $T05$, $T06$, $T08$ and $T09$. Based on this result, in the second phase, we try to minimize the total time of executing subtasks.

5.2 Second Phase

Let C_p denote the amount of required computation power, in terms of performance index, to execute a subtask attached to a transition t, and PI_i denote the performance index of node i. Moreover, let $S_{p,i}$ be a 0-1 (integer) variable that denotes the fact that place p is located at node i (the value is one) or not (the value is zero). Using these variables, we introduce the following formula for allocating places to nodes so that the total execution time is minimized

$$F_{min} = \sum_{i \in N} \sum_{p \in P} \frac{C_p}{PI_i} S_{p,i}$$

where P is the set of places and $\frac{C_p}{PI_i}$ is the time for node i to execute the subtask attached to place p. We apply the above to the considered example and obtain the place location table given in Table 5.

We have executed the obtained service and measured its performance. Fig. 11 shows the time chart and the performance. We observe that the response time is 99 seconds and it is less than half that obtained in the initial phase. Thus, the second phase, contributed significantly to obtaining an implementation with a better performance. However, there are still some problems in Fig. 11. The graph shows that the utilization of the overlay link from A and B is high but the utilization of other links is low. Therefore, we investigate the possibility of improving performance by utilizing other overlay links. In the initial and second phase, we have optimized the performance by minimizing the number of data exchanges and the execution time of subtasks. However, Fig. 11 shows that the time occupied by data transmissions may affect performance. Therefore, in the third phase, we try to adapt an optimization strategy that considers both transmission and execution times.

5.3 Third Phase

Let $D(X_{ij}^t)$ denote the size (amount) of data of a message X_{ij}^t (X is α or γ) and let $L_{i,j}$ denote the overlay link throughput from node i to node j, which can be obtained as described in the second

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

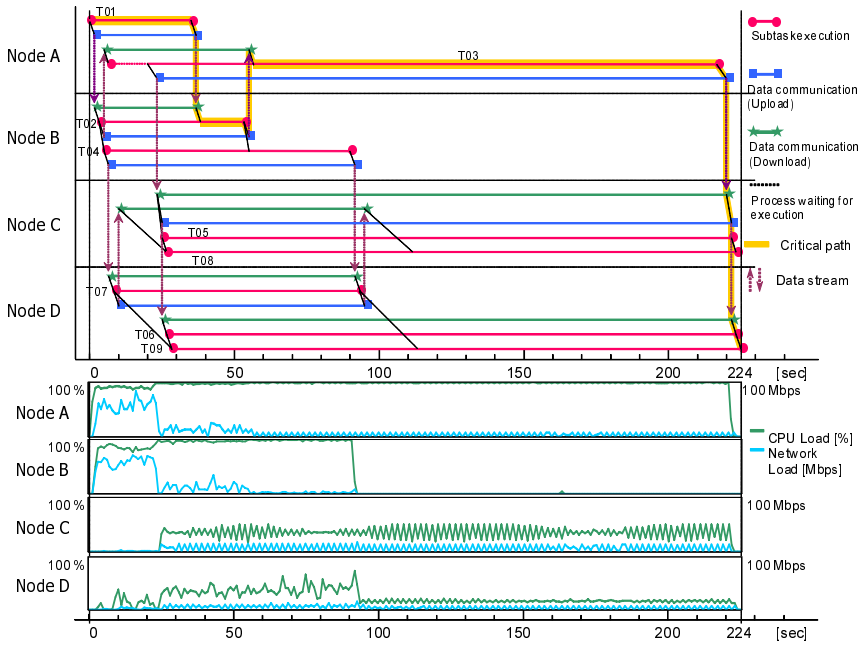

Figure 10: Timing chart of the service and node performance (in the initial phase).

Table 6: Place location table (in the third phase)

Node	Place Assignment
Node A	P08, P09, P11
Node B	P04, P06
Node C	P01, P02, P03, P05
Node D	P07, P10

phase. Using these values, we have use the following formula

$$F_{min} = \sum_{i \in N} \sum_{p \in P} \frac{C_p}{PI_i} S_{p,i}$$

$$+ \sum_{i,j \in N \times N (i \neq j)} \sum_{t \in T} \frac{D(\alpha_{ij}^t) * \alpha_{ij}^t + D(\gamma_{ij}^t) * \gamma_{ij}^t}{L_{i,j}}$$

to minimize the sum of the execution time and transmission time.

We apply the above to our application example and obtain the evaluation shown in Fig. 12. The response time is 84 seconds and thus by considering both the computation power and the traffic amount, we could obtain a better performance than only using the initial and second phases described above.

6. RELATED WORK

Recently, more and more computers are connected to the Internet and new distributed computing paradigms such as P2P computing [16], grid computing [9, 10], and service overlay [6, 17, 18] are arising. An important yet common concept to all these Internet-based distributed paradigms is the management of overlay networks. Since computers engaged in a network are usually heterogeneous and located on different network segments, we need to care about the network dynamics and the heterogeneity of nodes in designing a computation services of overlay networks. Furthermore, distributed programming is very complicated because we need to organize several programs to provide services taking into account the communication between these programs and services. To overcome some of these difficulties, in overlay networks, several computer-supported design methodologies have been considered [1, 11, 12, 15, 16].

iOverlay [11] provides a distributed platform on overlay networks. This is a middleware that has several features in implementing and evaluating overlay services. For example, the middleware has a mechanism for sending and receiving messages efficiently on overlay networks and this allows developers to concentrate on the development of applications on the overlay networks. Also, Arigatoni [1] helps in implementing distributed services by offering a resource discovery mechanism to find resources needed to execute services. Similarly researches like [12, 16] focus on alleviating developers' effort in designing distributed services. MACEDON [15] mainly focuses on helping the evaluation and analyses of distributed services executed on P2P networks by allowing to use both simulators and testbed networks like PlanetLab.

Compared with the above methodologies, in our work, we deal with services that are decoupled from overlay network specifications. A service can be written in a centralized way, using a known Petri net model, and developers can start building the design without being aware of overlay network architecture. The distributed version of the given service description is automatically derived to alleviate developers' load in writing distributed programs directly. Also, the resource optimizer allows optimizing the performance of the service using some given objective functions. Several objective functions are provided for the convenience of deployment of programs with better performance. A toolset that integrates our work with existing methods and tools is provided.

7. CONCLUSION

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

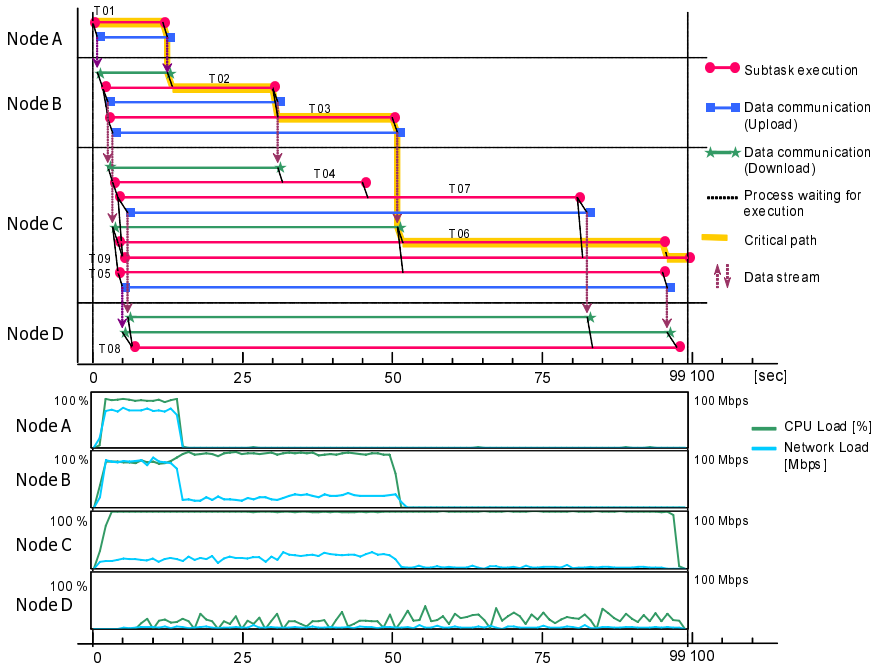

Figure 11: Timing chart of the service and node performance (in the second phase).

In this paper, we have proposed an integrated development environment for supporting the development of overlay services. In particular, we have used Pr/T-net and related tools for writing services descriptions in a centralized and network-independent way. Then, we have presented a tool for deriving a distributed version of the service taking into consideration the targeted overlay network specification. Our tool interprets the distributed specification and let the overlay nodes execute the assigned tasks of the specification. During that time, the tool monitors the utilization of overlay links and occupation of processors so that related information can be used for building a better implementation. Developers only write the service description and our tool supports the corresponding subsequent design and development tasks. An experimental study using a realistic application example is given. The study demonstrated the benefits of using our tool for providing a distributed implementation of a given service and for finding bottlenecks in the implementation by taking into account the given overlay network properties. A re-design that fits better the considered network can then be implemented without additional efforts from the developers side. Currently, we are working on opening our tool to the public. Furthermore, we are planning to extend the concept provided in this paper to other domains such as P2P networks of sensor gateways. In parallel, we would like to test the scalability of our tool on large-scale overlay networks such as PlanetLab.

8. ACKNOWLEDGMENTS

This work was supported in part by Research and Development Program of “Ubiquitous Service Platform” (2008), The Ministry of Internal Affairs and Communications, Japan.

9. REFERENCES

[1] D. Benza, M. Cosnard, L. Liquori, and M. Vesin. Arigatoni: A simple programmable overlay network. In *JVA ’06: Proceedings of the IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing*, pages 82–91. IEEE Computer Society, 2006.

[2] L. A. Cherkasova, V. E. Kotov, and T. Rokicki. On net modeling of industrial size concurrent systems. In *Proc. of 14th Int. Conf. on Application and Theory of Petri Nets 1993 (LNCS 691)*, pages 552–561. Springer-Verlag, 1993.

[3] J. de Figueiredo and L. Kristensen. Using coloured Petri nets to investigate behavioural and performance issues of TCP protocols. In *Proc. of 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN*, pages 21–40, 1999.

[4] H. J. Genrich and K. Lautenbach. The analysis of distributed systems by means of Predicate/transition-nets. In *Proc. of Int. Symp. on Semantics of Concurrent Computation (LNCS 70)*, pages 123–147, 1979.

[5] H. J. Genrich and K. Lautenbach. System modeling with high-level Petri nets. *Theoretical Computer Science*, 13(1):109–136, 1981.

[6] X. Gu, K. Nahrstedt, and B. Yu. Spidernet: An integrated peer-to-peer service composition framework. In *Proc. of IEEE Int. Symposium on High-Performance Distributed Computing (HPDC-13)*, 2004.

[7] P. Huber and V. Pinci. A formal executable specification of the ISDN basic rate interface. In *Proc. of 12th Int. Conf. on Application and Theory of Petri Nets*, pages 1–21, 1991.

[8] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use Volume 1: Basic Concepts.* Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1997.

[9] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science & Engineering*, 3(1):78–83, 2001.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580

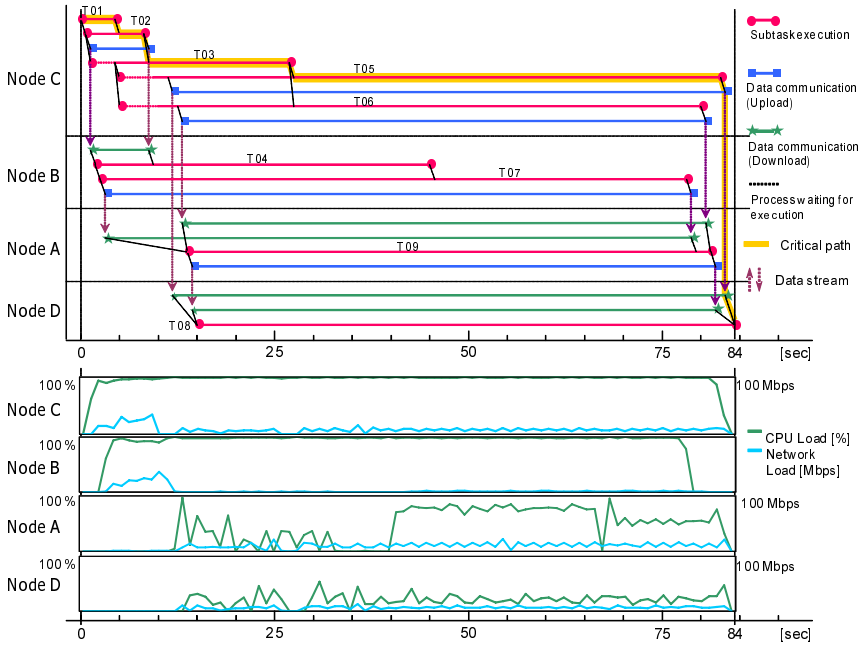

Figure 12: Timing chart of the service and node performance (in the third phase).

[10] S. M. Larson, C. D. Snow, M. R. Shirts, and V. S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[11] B. Li, J. Guo, and M. Wang. iOverlay: A lightweight middleware infrastructure for overlay application implementations. In *Proceedings of the Fifth ACM/IFIP/USENIX International Middleware Conference (Middleware 2004), also Lecture Notes in Computer Science*, pages 135–154, 2004.

[12] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, 39(5):75–90, December 2005.

[13] J. L. Rasmussen and M. Singh. Designing a security system by means of coloured Petri nets. In *Proc. of 17th Int. Conf. on Application and Theory of Petri Nets (LNCS 1091)*, pages 400–419, 1996.

[14] A. Ratzer, L. Wells, H. Lassen, M. Laursen, J. Qvortrup, M. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for editing, simulating, and analyzing coloured Petri nets. In *Proc. of 24th Int. Conf. on Application and Theory of Petri Nets*, 2003.

[15] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat. MACEDON: Methodology for automatically creating, evaluating, and designing overlay networks. In *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI2004*, pages 267–280, 2004.

[16] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay weaver: An overlay construction toolkit. *Computer Communications*, 31(2):402–412, February 2008.

[17] M. Wang, B. Li, and Z. Li. sFlow: Towards resource-efficient and agile service federation in service overlay networks. In *Proc. of 24th Int. Conf. on Distributed Computing Systems (ICDCS2004)*, 2004.

[18] D. Xu and K. Nahrstedt. Finding service paths in an overlay media service proxy network. In *Proc. of Int. Conf. on Multimedia Computing and Networking 2002 (MMCN2002)*, 2002.

[19] H. Yamaguchi, K. El-Fakih, A. Hiromori, and T. Higashino. A formal approach to design optimized multimedia service overlay. In *Proc. of 15th ACM Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2005)*, pages 57–62, 2005.

[20] H. Yamaguchi, K. El-Fakih, G. v. Bochmann, and T. Higashino. Deriving protocol specifications from service specifications written as predicate/transition-nets. *Computer Networks Journal*, 51(1):2581–284, Jan. 2007.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5580
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5580