ESPRIT PROJECT EP28425

Broadband Integrated Satellite Network Traffic Evaluations

Deliverable 1.2

Knowledge Based User Behaviour Modelling

Status / Version : **DELIVERABLE / FINAL**
Date : **September 30, 1999**
Distribution : **Public**
Code : **BISANTE/DEL12**
Author (s) : **G. Kotsis (Ed.)**

Abstract: In this deliverable we elaborate the proposed approach to characterise the behaviour of users in a networked environment with respect to their interaction with applications and services. Based on measurements, knowledge about user behaviour is derived and represented in an adequate way to be used by the models.

From the revised version, which is public, all case studies have been removed, as they contain sensitive data, which should be kept confidential within the consortium. Only two brief excerpts from the case studies are included illustrating the mapping of layers in a modelling study.

The BISANTE Consortium consists of :

Thomson-CSF Communications	Partner	France
Netway	Partner	Austria
Solinet	Partner	Germany
University of Vienna	Associated Partner	Austria
University of Surrey	Associated Partner	United Kingdom
Institut National des Télécommunications (INT)	Associated Partner	France

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

A knowledge based approach for user behaviour modelling to be used in network traffic simulation is proposed. The concept is based on a layered modelling framework, as presented in deliverable D1.1 of this project.

The motivation for a **layered approach** are twofold. On the one hand, a layered approach allows for a description of the traffic at various levels from the **modelling point of view**. In particular, we identify a top-level, user behaviour oriented point of view, the workload generator level, which translates down to the lowest level, the service level, at which we can characterise the actual type of traffic from a network-oriented point of view. On the other hand, **aggregation techniques** can be applied to make the models more tractable from an **evaluation point of view**.

Both, the **modelling framework**, which identifies the layers and entities at each layer, as well as the **methodology**, which describes the usage of the framework, will be presented.

The notion of **knowledge based** user behaviour modelling refers to the concept of providing a **library of models and parameters** at each layer to support reusability and ease of use of the approach. At each layer, a set of predefined models is defined which incorporates existing knowledge on user behaviour. As the models are parametrisable and extensible, this existing knowledge can also be modified according to new insights gained.

Instead of being either restricted to a set of (typically simplified) traffic source generators available in the network simulator of the modeller´s choice, or having to develop complicated source models from scratch, the BISANTE approach aims at providing the modeller with an extensible set of modules for traffic modelling at different layers from which he or she can choose to construct the desired user behaviour. In addition, user behaviour profiles, which are ready-to-be-used combinations of models and parameters describing a particular instance of user behaviour, will be provided. Thus the modeller can freely choose between the whole spectrum of either using a fully-specified profile, or constructing and integrating his/her own modules into the framework. Well defined interfaces among the layers will guarantee interoperability among layers.

In Deliverable DEL12-CS, the **four case studies** identified in DEL11 and detailed in DEL11-CS will be used to demonstrate the methodology.

In case study one, modelling **HTTP traffic**, the layered approach will be applied to translate from user behaviour in terms of opening web session through a web browser to the actual load in terms of IP packets submitted to the network. This should demonstrate the **ability to modify user behaviour at a high level** (e.g. increasing the arrival rate of user sessions) and to study the effect on network load, a question which is of particular importance for NETWAY, the internet service provider.

In case study two **audio/video broadcasting** will be investigated. For this case study, the crucial point is in modelling **the appropriate type of traffic at** the lowest level, the **service layer**. The layers atop are merely responsible for starting/terminating a broadcasting stream. Initial results are given on the actual type of model used at the service layer, a detailed description of all models and related parameters (which will be referred to as user behaviour profiles) will be the focus of DEL13.

In case study three on **CSCW environments**, a class of applications is studied, which is interesting because it allows for **user interaction at various levels**. In this case study also some aspects of **mobile users** participating in a CSCW session are considered.

The fourth case study, which has been chosen, models a **company´s intranet**. We identify a subset of applications, which have been ranked as most relevant by THOMSON, one of the end users in the partner consortium, that we want to model. The challenge here is to adequately mimic the real behaviour of users in the intranet in terms of specifying a **representative mix of usage of applications**.

The next steps are

- to further elaborate the case studies in terms of constructing the models (to be reported in DEL13)
- to give a detailed specification of the types of models that will be implemented (to be reported in DEL31) and,
- to perform simulations and measurements for validation of the models (to be reported in DEL42 and DEL43).

2. OUTLINE

The objective of this deliverable is to further elaborate the layered approach for network traffic modelling as described in DEL11, to represent the methodology in detail, to introduce the concept of knowledge based user behaviour modelling and to demonstrate the methodology on selected case studies.

In **Section 3**, we will recall the layered framework presented in DEL11, which was adopted to best meet the requirements of the knowledge based user behaviour approach. It was necessary to explicitly represent the workload generator associated with a network node (set) at the top level.

In **Section 4** we discuss how to include knowledge on user behaviour into the framework.

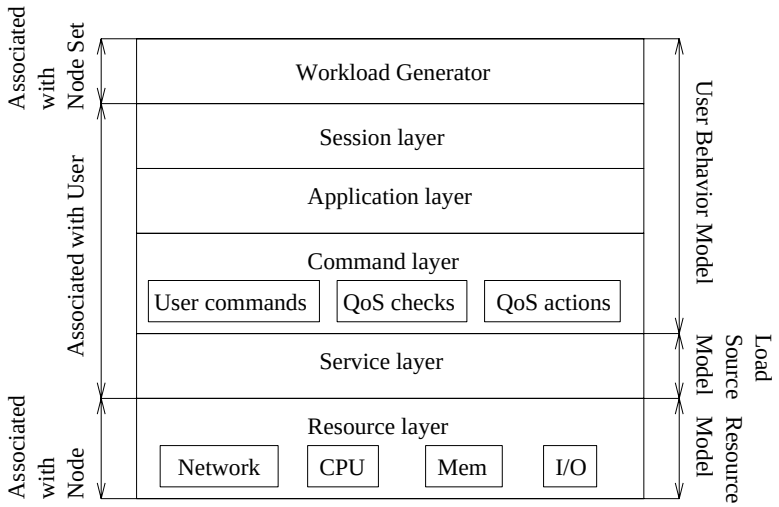
In **Section 5** we discuss aggregation techniques.

In **Section 6** we briefly give two examples on how to represent traffic models following the layered approach. More details on the examples are given in a separate Deliverable, DEL12-CS.

The conclusions (**Section 7**) summarise the results presented in the deliverable and give an outlook on the next deliverable.

3. MODELLING FRAMEWORK

In this section, a layered approach for characterising network load is described. The goal is to find a set of layered models that can be plugged onto each other to represent some typical user behaviour that has been either observed, or that is projected to be seen in the future. The models will be created top down, starting always at the highest layer and then going down to the chosen level of detail, each layer adding another level of detail to the model.



At the top level, the **workload generator** is responsible for managing the activities of a (set of) user(s) of the network. A model has to represent the inter-arrival times of users as well as the type of applications a user would work with.

The workload generator is responsible for triggering activities at the next lower level, the **session layer**. A session is defined as the activities of a particular user in terms of starting/stopping applications.

At **the application layer**, the model has to specify, how a user interacts with the chosen application, basically characterising the sequence of commands he or she issues.

At the **command layer**, a distinction has to be made between explicit user commands and (periodic or event triggered) quality of service checks and actions inherent to the application (examples of which will be given below).

At the **service layer**, a mapping is made from the commands to the actual service generating load (in terms of bytes) on the network, but also vice-versa to be able to model feedback of network load to user behaviour.

Finally, the **resource layer** describes the physical node resources.

We refer to the first four layers as the **user behaviour model**, while the service layer is the actual **load source model**.

The framework is designed in a way, that what to do next is passed from the highest layer down to the lowest by triggering actions. If such an action is triggered for layer i by the next higher layer i+1, layer i starts triggering a stream of actions for the next lower layer i-1 and waits for results from it. Results are then processed and, if necessary, passed on to layer i+1. The lowest layer in this stack generates the workload for the resource under observation. If the modeller chooses to skip intermediate layers, e.g. because the system under study does not require a model at that level of detail, those layers will be substituted by dummy layers, which simply pass on the actions.

The more layer models are plugged-in, the more detailed will be the workload description and the more accurate will be the result, leading also to more events to be simulated, increasing the simulator complexity, both with respect to CPU time consumed as well as memory requirements.

3.1. SIMULATION EVENTS

The simulation environment is assumed to work in an event-driven fashion. Events denote a piece of atomic work that is scheduled to happen at a specified virtual time point t_i. The order of events is assumed to be preserved, i.e. if there are two events e_1 and e_2, scheduled at $t_1 < t_2$, then e_1 will be executed before e_2. Thus, there must be either some global ordered list of events, or some mechanism must be able to unroll already executed events, in case an out-of-order execution has been detected.

Generating events of any kind is modelled by the inter-arrival time of such events and the type of the event itself. At each layer, thus a stream of events is produced. The stream of events can be influenced by results from lower levels (observed quality of service), be stopped by itself (for instance, a service may stop itself) or be started by higher layers. If a layer decides to stop itself, this must be passed to the next higher layer, which will then decide what to do next.

3.2. SIMULATION ARCHITECTURE

The simulation architecture is assumed to consist of the following parts:

- **Simulation kernel**: Contains the functionality to schedule and execute events.
- **User models**: Here, all layer models are defined. Within the framework, layer models of a certain type will be treated like model classes. At run-time, instances of these classes will be created and linked together.
- **Terminal equipment** or nodes: These denote the hardware that allows users to access the environment and to create workload.

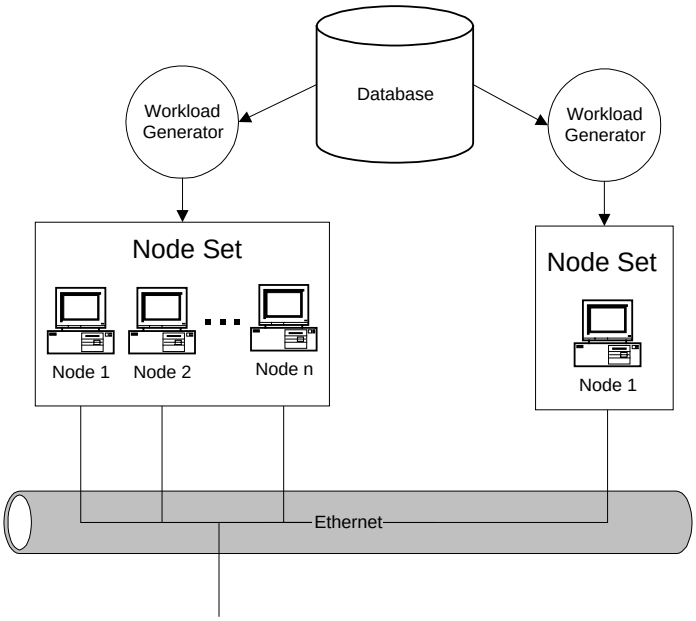
At simulation start, the workload generators will create user sessions and will place them to one of the available nodes. The user sessions will then start applications and thus create workload, which will be passed on to the resource layer model associated with the node.

3.3. TERMINAL EQUIPMENT

Terminal equipment or nodes denote the **hardware that applications run on**. They are also members of the network topology under observation. Terminal equipment is either created and connected to the network topology before simulation start, or is defined as a template. Template instances are then created at run time and are dynamically connected to the network topology at some predefined nodes (dial-in nodes). Terminal equipment can also move around and might be passed from one dial-in node to the other (hand-over).
Each terminal equipment must be associated with a resource layer model.

3.4. NODE SETS

Node sets consist of one or more terminal equipment. **Static** node sets consist of pre-defined nodes that are instantiated at simulation start. **Dynamic** node sets consist of node templates that are instantiated at run-time as the simulation evolves. Each workload generator must be associated with either one node or with one node set, choosing one of the nodes to host the next user session at run time.



3.5. LAYER SIGNALS

The idea behind the framework is to encapsulate the type of model and to define interaction among the models in terms of a standardised communication interface. This allows for construct libraries holding models of any type. For example, a Markov chain could be used to model the application layer, while the command layer could be modelled by a Poisson process. When actually performing simulation, the user should be able to choose any model for the various layers without being forced to know about the internal model details.

The different layers though must be able to **communicate** with each other. From an abstract point of view, this can be achieved by sending signals from one layer to the other. A set of signals thus must be defined for each layer that is used for communication. Each layer model then must be able to deal with such a set of signals, either by doing nothing, sending a signal to another layer or by scheduling one or more events.

At the beginning, a model instance has to be created (initialisation), but is still inactive. Once started, a model instance will become active in sending signals and/or generating events for simulation. Such an activity can be stopped with the possibility of becoming active again, and finally the model instance can be destroyed.

The corresponding signals causing the change of state of a model instance and notifying other instances of such a state, that have to be interpreted by all layers are:

- SIG_INIT, which will tell a model instance to initialise itself,
- SIG_START, which will tell the model instance to start its activity,
- SIG_STOP, which will tell a running model instance to stop its activities,
- SIG_STOPPED, which will indicate that a model instance has terminated its activity, and
- SIG_END, which will tell a model instance to terminate/destroy itself.

In addition, the service layer and the resource layer must be able to handle the following signals

- SIG_REQ, SIG_SEND, SIG_RECV,

which indicate the actual data transfer.

Signals can be transported from instance to instance by various means, depending on the programming language used for implementation and the underlying simulator kernel. An object-oriented implementation might choose member functions that are called either directly by other instances or by a central scheduler, as, for example, is done within ns.
Other simulator kernels will allow messages that are passed from one entity to the other, like, for example, PARSEC and OMNeT++. If one is interested to create layer models that can be run on different simulator kernels, then some simulator specific

layer between the models and the kernel must be inserted, though this will not be treated in this work.

The following subsections will describe the purpose of each layer in detail. Many of the models will consist of a finite (yet dynamically changing) set of states, which might change over time.

3.5.1. WORKLOAD GENERATOR

At the highest layer in the framework, a workload generator is associated with each node set. This workload generator **creates user sessions** according to the desired statistical distribution. This can be done in two ways:

- The workload generator is associated with a **static** node set. In this case, a currently free member of the static node set is chosen to host the next user session. As an example, a company network can be taken, where employees arrive in the morning and start generating network requests on their workstations.

- The workload generator is associated with a **dynamic** node set: In this case, nodes are created according to their node templates and are connected to a member of their associated node sets (dial-in nodes). Here, as an example, a mobile communication network can be taken, where users arrive according to some arrival rate, and start generating workload using their mobile terminal equipment.

Generating user sessions can either be done synchronously or asynchronously to its next lower layer:

- Synchronous: The time to create the next user session instance is computed only, if the next lower layer has ended the session. This results in a sequence of none-overlapping sessions.
- Asynchronous: The time to create the next user session instance is computed right after the previous user session instance has been created. In this case, sessions can overlap and competition for the available nodes might occur.

Workload generators thus must be able to host one or more user sessions. The workload generator instances are created at simulation start. Each user session instance is then created and linked to its workload generator instance at run-time.

The following **signals** will be sent/received:

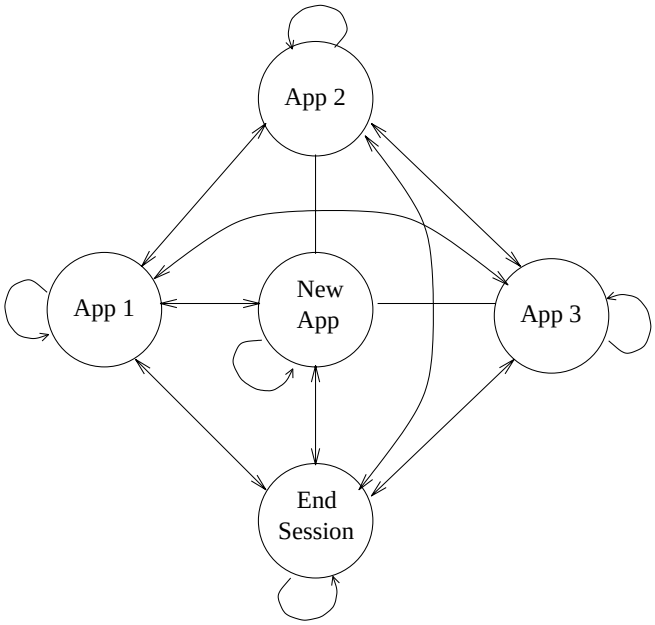
- received
  - from the outside world
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END
  - from the next lower Layer (session layer)
    - SIG_STOPPED
- sent
  - to the next lower layer (session layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END

3.5.2. SESSION LAYER

Models at this layer have the following tasks:

- Start applications.
- Choose the application to use.
- End session.

Note that it is only the application to use, not the service that is chosen. Also, the number of possible states of models at this layer may vary, as in principle, many applications can be started and stopped again.



Once, an application is chosen, the next lower layer starts services of this application. Starting and choosing applications can again be done either synchronously or asynchronously.

Session layer models are plugged directly on top of application layer models. Note that session layer models can be very simple dummy models that just start an

application (which in turn just chooses and starts a command sequence). This way, each TE/user can create workload with very small effort at a low level of detail.

The following **signals** will be sent/received:

- received
  - from the higher layer (workload generator)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END
  - from the lower Layer (application layer)
    - SIG_STOPPED
- sent
  - to the higher layer (workload generator)
    - SIG_STOPPED
  - to the lower layer (application layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END

3.5.3. APPLICATION LAYER

Application models define, how the user interacts with the chosen application. Basically, the user can perform one of the following interactions:

- Start a new command sequence.
- Choose a running command sequence.
- Stop the application.

If the application is stopped, the user session instance above must start another application or choose another one. The generation or selection of command sequences can again be done synchronously or asynchronously to the user commands at the next lower layer.

Application layer models are plugged on top of command layer models. Note that application layer models can be very simple dummies, having nothing more to do than starting a command sequence.

The following **signals** will be sent/received:

- received
  - from the higher layer (session layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END
  - from the lower Layer (command layer)
    - SIG_STOPPED
- sent
  - to the higher layer (session layer)
    - SIG_STOPPED
  - to the lower layer (command layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END

3.5.4. COMMAND LAYER

At this layer, sequences of commands for lower level services are generated. This layer consists of three independent sub-models.

3.5.4.1. USER COMMANDS

Here, services are started or running services are selected, and sequences of commands are issued for running services. Commands for running services denote the changing of parameters like the size or colour depth of a running video conference.

Amongst the possible commands are:

- Start a service
- Change service parameters.
- Stop the service.

If a service is started, it generates workload requests and passes them down to the resource layer. The observed quality of service (QoS) is part of the service and can be obtained by the next higher level. Services can stop themselves after delivering the result (for example HTTP). If a service is stopped, the command layer model must take over again and generate the next command.

User commands can be simple dummy models, which either do nothing or just start a service or stop it after some time.

3.5.4.2. QOS CHECKS

As long as a service is running, it delivers QoS descriptions. For each QoS description, a QoS level can be defined. QoS checks compare the observed QoS level to the requested QoS level. QoS checks can be performed as single or periodic checks:

- Single: The check is performed only once, after a time out occurred. If the service has not been completed, the appropriate QoS action is triggered (for example the requested web document has not been downloaded)
- Periodic. The check is performed periodically, until the service stops.

3.5.4.3. QOS ACTIONS

If QoS checks fail, an appropriate QoS action is triggered and performed. QoS actions and user actions are at the same level. They can start or stop services, or change parameters for running services.

At the command layer, the following **signals** will be sent/received:

- received
  - from the higher layer (application layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END
  - from the lower Layer (service layer)
    - SIG_STOPPED
    - SIG_RECV, which indicates that the service is delivering results
- sent
  - to the higher layer (application layer)
    - SIG_STOPPED
  - to the lower layer (service layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END

3.5.5. SERVICE LAYER

Service layer models consist of parameterised traffic generators (See Deliverable 1.1). Some parameters can be changed by higher level models and influence the QoS level that the higher level model chooses. Other parameters are fixed and describe a situation that can not be changed by users, like the distribution of file sizes at a web site.

Services can stop themselves, it they have performed their task (like delivered the web document, played a video). Services can also be represented by very simple models, which just create low detail workload by using simple fluid models or renewal models like Poisson arrivals.

The following **signals** will be sent/received:

- received
  - from the higher layer (command layer)
    - SIG_INIT, SIG_START, SIG_STOP, SIG_END
  - from the lower Layer (resource layer)
    - SIG_RECV, receiving n bytes from a resource
- sent
  - to the higher layer (command layer)
    - SIG_STOPPED
    - SIG_RECV, indicates, that the service delivers results
  - to the lower layer (resource layer)
    - SIG_SEND, sends n bytes to a resource
    - SIG_REQ, request n bytes from a resource

3.5.6. RESOURCE LAYER

Resource layer models describe physical node resources:

- Network (TCP/IP, ATM)
- CPU queue
- Set of memory pages
- I/O resources like disks

Each of these models must be implemented at least as a dummy resource. There is only one resource layer model instance per workload generator instance. The network resource model will in general be attached to another network resource model within the network.

When using existing simulation kernels like ns, some of these resources will only be entry points to the network simulator kernel.

At the resource layer, the following **signals** will be sent/received:

- received
  - from the higher layer (service layer)
    - SIG_SEND: A service sent n bytes to a resource (network, I/O)
    - SIG_REQ: A service requests n seconds or n bytes from a resource (CPU, main memory, I/O)
- sent
  - to the higher layer (service layer)
    - SIG_RECV, passing n bytes to a service

4. THE MODEL DATABASE

The core of the approach is a database of user behaviour and traffic models to be reused either with existing parameters or with new parameters specified by the user of the BISANTE simulation workbench.

4.1. TERMINOLOGY

Let us introduce the following terminology:

The Layer Model is an instance of a particular modelling formalism applied at a particular layer in the modelling framework.

A generic model at layer i will simply model two aspects: namely the interarrival time of events and the associated type of event as defined at layer i. Models for layer i must be able to deal with incoming signals at this layer and must be able to generate signals as necessary. For every layer (workload generator, session, application, command, and service), explicit models of any type can be developed and implemented.

A model at one layer will consist of three parts:

1. Model type: This is a unique identifier for each model type at each level

2. Model implementation: This will be done in the programming language of choice resulting in a modelling module

3. Model parameters: Models must be able to accept different parameters telling the model instance how to behave.

The Model Skeleton is composed of a hierarchy of models covering layers i, i+1, i+2, ... A complete skeleton covers all layers and is the basis for a user profile.

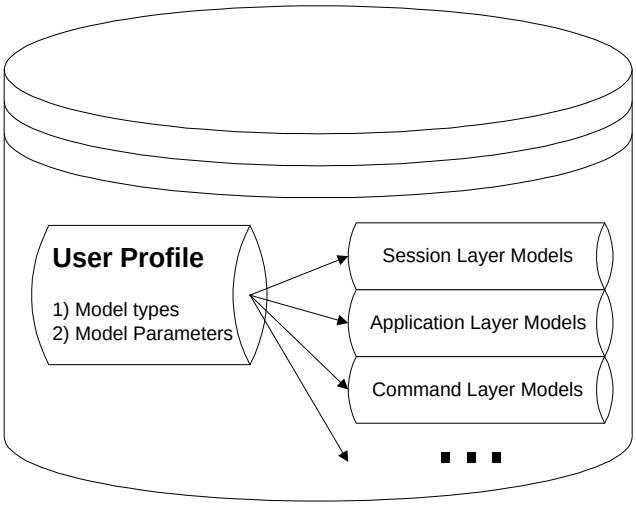
The User Profile User profiles will provide pre-defined sets of user models and model parameters covering all layers. They will therefore contain

- A set of model types for each layer (complete skeleton)
- Values for all model parameters

Instead of choosing model types for each layer, the modeller thus will be able to simply choose a fully specified user behaviour profile for simulation runs.

Note, that a user profile is always defined for a specific class of users. For example, there might exist a profile for the “typical web surfer”, describing at

the top level the rate at which users start new web-sessions, at the session level the types of applications (different web-browser), and the commands available within a web browser (which may depend on the chosen browser), including for example start/interrupt an http session or an ftp session, until finally at the service level the appropriate traffic source generator models are generating the stream of packets to be transmitted over the network.



The Model Database is a database holding all types of models and necessary model parameters. The analyst working with the BISANTE workbench will be able to choose any model available for each layer and will be able to specify and experiment with the model parameters. Also the user profiles are stored in the database for convenient access.

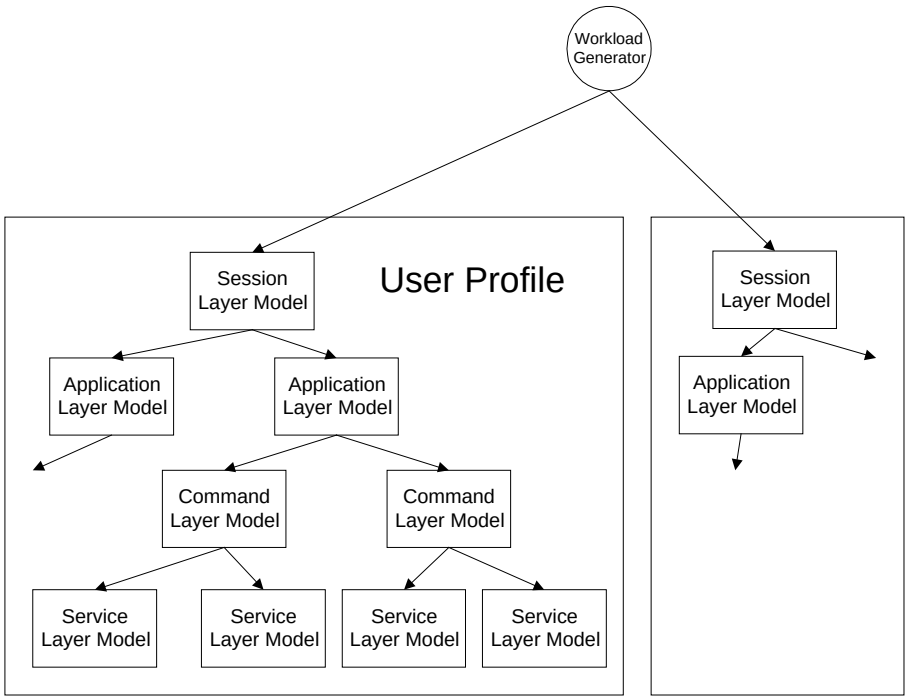
Within the BISANTE project, this database will be designed conceptually only. It is not the purpose of the project to develop a fully operable database or to design the interface for accessing the models in the database.

4.2. THE MODEL HIERARCHY

Models of layer i will be able to create one or more model types at layer i-1. This results in a tree-like data structure of models instances describing the user behaviour.

The user profile will be described by a static tree structure. There, each layer model will be able to construct a set of models at layer i-1. This information defines the profile and does not change during run-time.

At run-time, model instances will be created dynamically, and this information will be described by a dynamically changing tree structure.



5. AGGREGATION SCHEMES

If the simulation scenario gets too large, the simulation time needed to compute the desired performance metrics will also get too large. One way to get around this drawback is to apply aggregation techniques.

To understand the aggregation schemes proposed in BISANTE, let us recall, that the BISANTE workload generators are user centric in the way that the workload always depends on the number N of the currently active user sessions. In the purest form, each active session will be represented by its own session profile, consisting of the user model layers (each being represented by a certain type of stochastic process) and the corresponding process parameters. Once started, the stochastic processes will eventually create network load.

This approach requires process events being created, scheduled and computed at each layer of all model instances. Though this approach is the most accurate one, in large scale simulations of thousands of users there will be a too large overhead of simulation events involved. In such scenarios, only averaged network workloads are important, singular traffic peaks and single packets will have no significant influence on the derived performance metrics.

Therefore, the BISANTE workload generators will include a flexible aggregation scheme, presenting not only one or two aggregated model types, but merely a framework for aggregation, allowing to choose that level of aggregation that, while preserving a sufficient accurate representation of the observed network load, generates workload with least overhead.

The BISANTE aggregation scheme will start at the user models for one session and will have the same layered structure. Like in models for one user only, in each layer, a stochastic process will generate events and will start and stop processes at lower levels. At the service layer, the appropriate model will finally produce network load.

The aggregation scheme will then allow to create aggregated versions of the model for one user by following one or several of the three aggregation dimensions. The further is proceeded along one of the three dimensions, the higher will be the aggregation, i.e. more and more events will be represented by one event only. The scheme can be viewed as a three-dimensional cube, where, starting at the upper, left edge, one can proceed towards the edge at the opposite side of the cube. This edge then represents the highest possible degree of aggregation. The whole scheme can be seen in Figure 1.

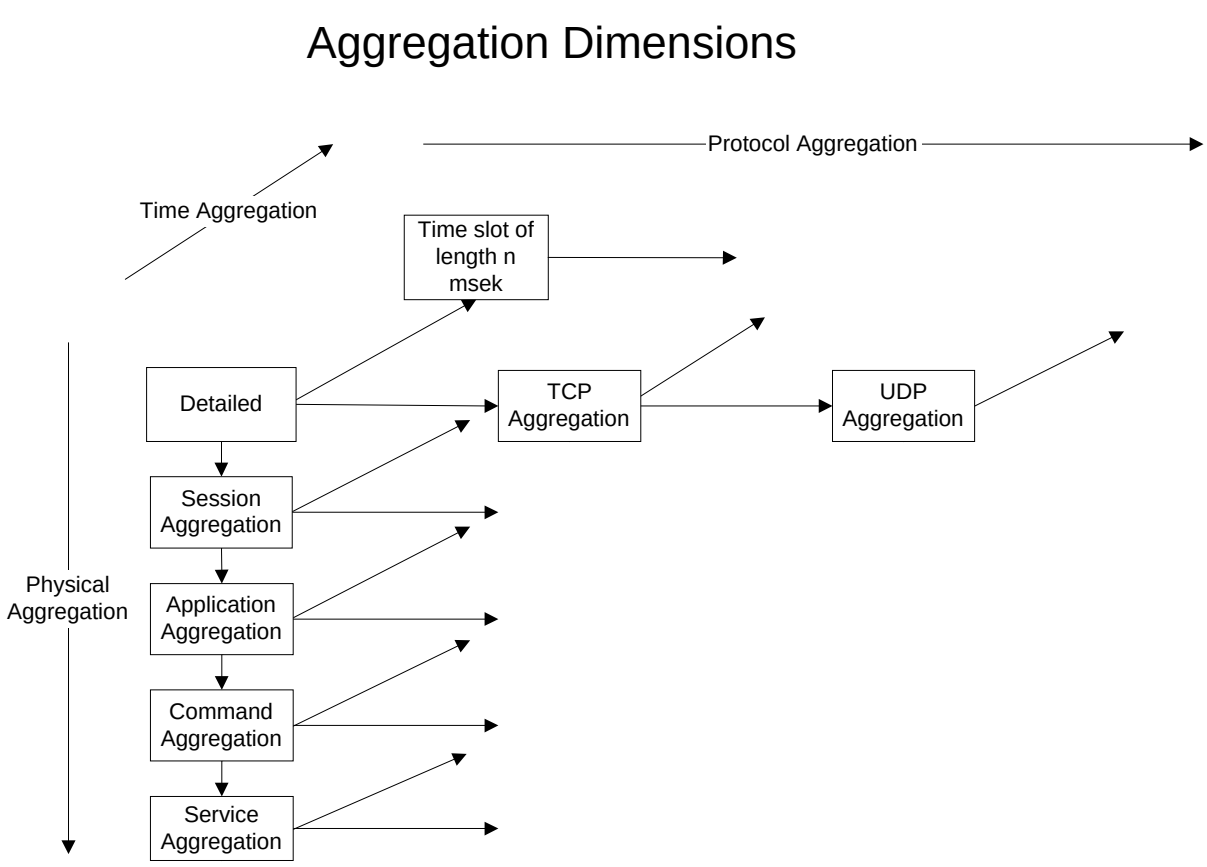


Figure 1: Aggregation scheme.

It is important to understand that the three aggregation dimensions are independent of each other, i.e. one can choose any combination of degrees of aggregation of them. The following subsections will define the three aggregation dimensions as well the procedure necessary to create such an aggregated model.

5.1. PHYSICAL AGGREGATION

Physical aggregation means aggregating the network workload of N users into one model instance. As can be seen in Figure 1, physical aggregation can be done at each layer of the layered user models. In order to understand the idea of physical aggregation, it is important to understand what is happening at each layer of a layered user model. At each of such layer, a stochastic process will be started and will generate events at a rate that has been observed from real user sessions. Aggregating at layer x now means exchanging the normal process at layer x with one that has a higher event generation rate, as can be seen at N users instead of just one. It also means that all higher layers will be represented by dummy layers only, which just will create and start lower layers.

The idea now is that by using such an aggregation scheme, all simulation events at **higher** layers will be saved, whereas no events at lower layers will be saved. The further down one proceeds, the more layer levels will be aggregated and thus, the more events will be saved.

Models at lower layers will still behave as if having been created by a user model representing one session only. This way, available models for this layer can be reused without constructing new stochastic processes.

5.1.1. SESSION AGGREGATION

In session aggregation, the session layer will be driven by a stochastic process creating applications of different types as before, but this time with a rate similar to that of N users. It follows that one important input parameter of such a process will be the number of users N.

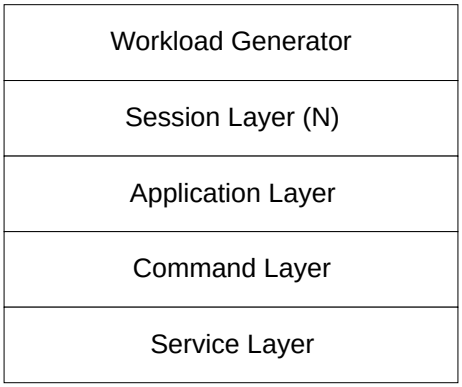


Figure 2: Session aggregation.

As can bee seen in Figure 2, the Workload Generator will still create user sessions. What is new is the fact that the session layer model now represents the process of creating applications that would be observed from N users instead of one.

This way, the Workload Generator will save N-1 times the creation of sub-layers, sending initialisation and start signals and so on.

5.1.2. APPLICATION AGGREGATION

In application aggregation, the session layer will be represented with a dummy layer, i.e. it will only produce one event, the creation and starting of an N-aggregated application layer model. This application layer model will represent a number of different applications as being desired in the user profile. Such an aggregated application model can, for instance, represent web browsers, e-mail clients and video streaming applications. With a rate as would be observed by N users, this model will create a stream of command sequences of different types being possible in the represented applications.

The whole scheme can be seen in Figure 3.

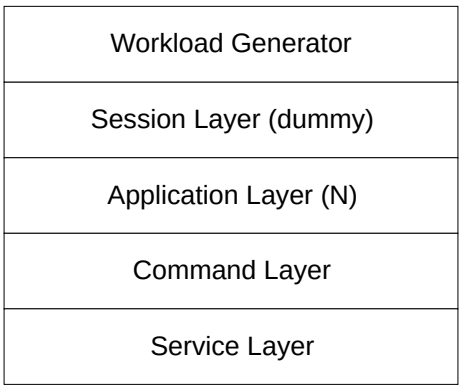


Figure 3: Application aggregation.

Once started, the command sequences will behave as if they had been started by a model representing one single user. The command sequences will start a certain service and will start producing a stream of commands for it, possibly only one further for stopping the service, or no one, if the service is to stop itself.

5.1.3. COMMAND AGGREGATION

Right now, there is no scheme for aggregating at the command layer. This aggregation step will thus be omitted.

5.1.4. SERVICE AGGREGATION

A newly started service will open and close TCP connections or will send UDP packets to other hosts. Aggregation at the service layer thus means starting one service that will produce the same amount of network load as being observed for N users.

All other layers will be represented by dummy models.

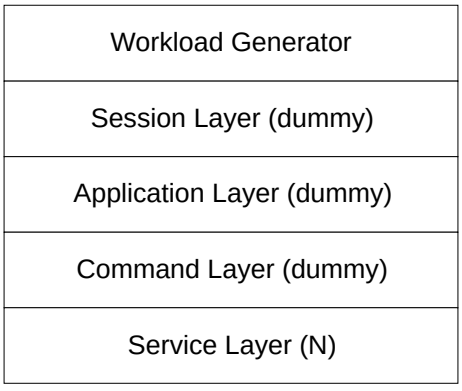


Figure 4: Service aggregation.

5.1.5. REPRESENTING N USERS

The question arises as how to represent exactly N users using the above approach. In order to do this, we define the following situations:

1. The arrival times are produced by Poisson processes. In this case, the arrival rates can be simply multiplied by N.
2. The arrival process can not be modelled by a Poisson process (The inter-arrival times are not exponentially/geometrically distributed). In this case, separate processes will be modelled for 1, 10, 100, 1000, and so on users. An N-user model will then be represented by the decimal equivalent of N. If, for example, 1387 users are to be modelled, then workload will be modelled with one N=1000 model, three N=100 models, eight N=10 models and 7 detailed (N=1) models. This way, only a small number of aggregated models at this aggregation level have to be created, whereas during runtime, the number of dynamically created models representing the workload is kept small also.

Generally, the Workload Generator will be responsible for creating the right mix of aggregated models to represent N users.

Creating aggregated models themselves will be explained in section 5.4.

5.2. PROTOCOL AGGREGATION

Many services such as SMTP, FTP, TELNET, POP3, NNTP and so on will use TCP as a reliable means for transportation. When simulating TCP, the simulator in principle has to reflect the real protocol procedures itself. Each TCP stream must be opened by sending three packets, where sender and receiver agree on the size of the send window. Then, the sender must wait for acknowledgement packets, whereas the receiver must send these packets accordingly. Both must keep track of such packets, must count the sent bytes, keep a list of out-of-order packets and acknowledgement packets. Upon receiving no or duplicated packets, some kind flow control mechanism will be activated that scales down the send window size and so on. Basically, for each TCP connection, there is a significant amount of control overhead involved, even if the TCP connection is only simulated.

Protocol aggregation now tries to drop this TCP overhead by changing to UDP. For this aggregation scheme, there are two possible steps:

1. TCP aggregation: In this aggregation step, the sending service still opens a virtual TCP connection. The traffic on this stream thus is emulated by an associated stochastic process sending UDP packets. The size and inter-arrival times of the UDP packets must mimic the inter-arrival times and sizes of the observed TCP packets. As long as this virtual TCP connection is open, a process will create packets for it and will send them as UDP packets. At the other end of this virtual TCP connection, a stochastic process will send UDP answer packets. Once, the service decides to close the connection, the associated process will stop producing packets.
2. UDP aggregation: The services no longer open or close even virtual TCP connections. All packets are sent as UDP packets, mimicking the inter-arrival times and packet sizes of all observed TCP and UDP packets.

Protocol aggregation aims on saving simulation overhead of TCP connections. While the first aggregation step will naturally preserve some auto-correlation by using an On-Off scheme, the second can only use special stochastic processes to do this.

5.3. TIME AGGREGATION

All stochastic processes so far have been assumed to consist of continuous point processes, where the inter-arrival times of events are drawn from continuous (though not necessarily independent) distributions. The more events are produced, the smaller these inter-arrival times will be. Especially, if many small packets are sent very often, each individual packet will contribute only little information to the overall simulation result and thus it is thinkable of instead of sending several small packets to send one large packet being the sum of all these small packets.

Time aggregation is the generalisation of this idea. The aggregated processes are represented by point processes with a discrete inter-arrival time distribution. For such a process, time is divided into time slices of the same size, say K milliseconds. The process will then decide, how many time slots to advance, until the arrival of a new event. Upon its arrival, the next observed value of this process (number of created sessions, number of opened applications, number of sent bytes) will be drawn from a suitable (auto-correlated) stochastic process.

This way, all events of a process happening inside the same time slot can be aggregated into one single event, and thus, simulation time can be saved.

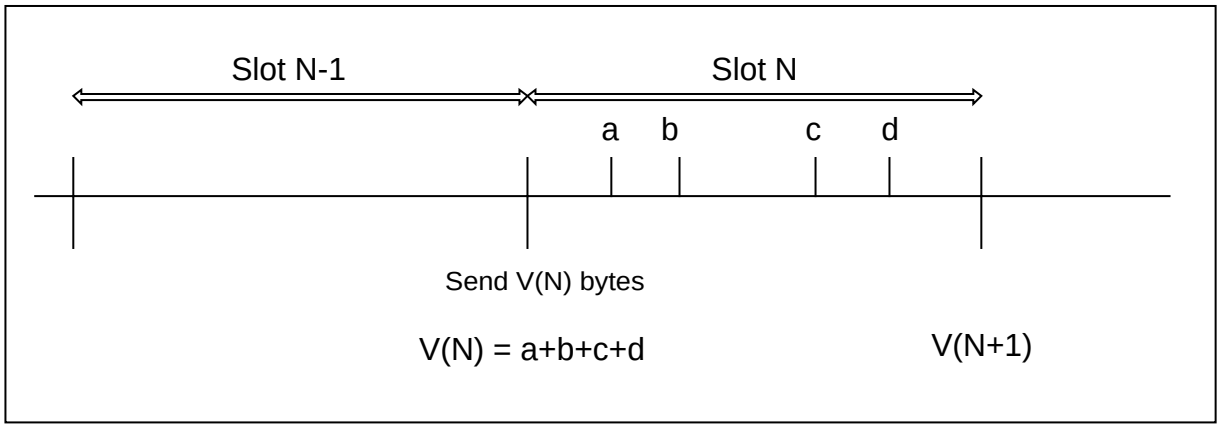


Figure 5 : Time aggregation: The four events a, b, c and d are aggregated into one event N.

One model layer may also hold several processes interacting with each other. For example, at the service layer, one aggregated process might compute the next number of byes to send, while another computes the number of packets to be used. In this case, the bytes would be distributed evenly amongst these packets.

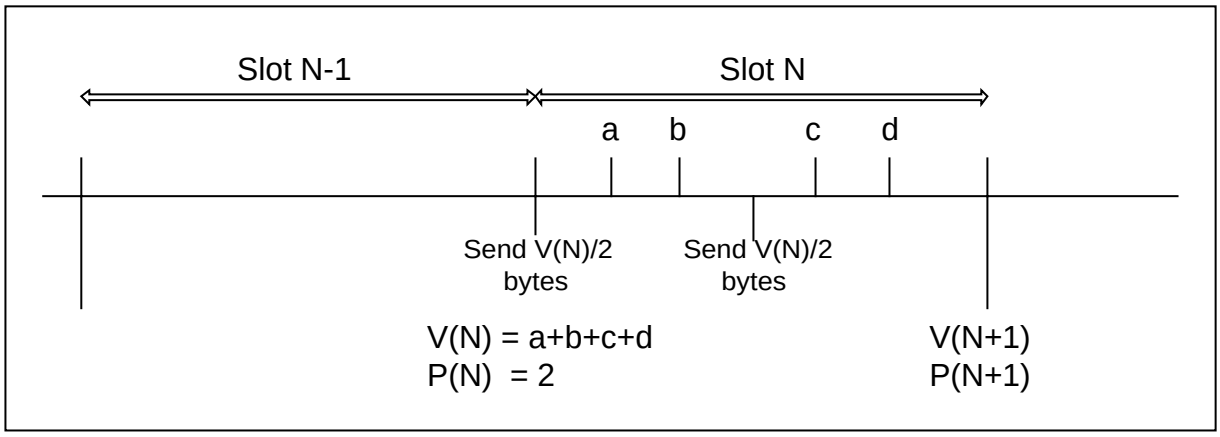


Figure 6 : Two synchronised time aggregated processes. One computes the number of bytes to send in slot N, the other computes the number of packets sent in slot N.

Time aggregation can be used on any model layer, starting at the Workload Generator until the Service Layer. The aggregation can be done for one layer only, or even for one layer of a particular user class. All other processes can still use continuous time distributions. It is just a matter of replacing one special layer model with another.

Time aggregation is also suitable, if the observed traffic shows no autocorrelation at the packet level, but at some time-aggregated level.

5.4. BUILDING AGGREGATED MODELS

Aggregated models will be based on available detailed user models. First it will be decided, which dimension is used for aggregation.

5.4.1. PHYSICAL AGGREGATION

For physical aggregation, the layer level for aggregation will be chosen first. Higher layers will be represented be simple dummy models, lower layers will be the same as in the detailed model.

In order to create appropriate aggregated layer model, a simulation tools will be used (OPNET or ns), where exactly N=10,100,1000,10000 and so on detailed users will be simulated. This simulation will yield the stochastic behaviour of the aggregated model. The traced data will then be used to build an appropriate stochastic process for N users at the desired layer level. For instance, at the session layer, the aggregated process will simply open many more applications per time unit as observed by the simulation.

5.4.2. PROTOCOL AGGREGATION

In this aggregation dimension, again a number of detailed user models will be simulated and traced. Here, any number N will be fine, as long as it yields sufficiently large data sets.

- For TCP aggregation, each individual running service will be observed. For each service class, first On-Off processes will be created, representing the opening and closing of TCP connections. Then, the point process generating the TCP packets will be observed and modelled. The aggregated point process will contain also

TCP control packets and control data. The model will then use UDP packets to simulate the observed TCP connections.

- For UDP aggregation, all observed UDP and TCP packets will be modelled by one point process per service. No more virtual TCP connections will be opened.

5.5. TIME AGGREGATION

First, the process to aggregated will be defined. For instance, the service *Video Stream* of a particular user class. Again, a large number of users will then be observed. The observed services will produce a sequence of time points and observed values that will be stored in trace files. This data can then be time aggregated and modelled with a discrete time stochastic process. The resulting process, for example for the video stream, will then produce UDP packets only on certain time points. The sizes of these packets will be drawn from an appropriate distribution reflecting the sum of packets observed in the time slots.

In this example, an additional process might be created to model the number of packets sent in each time slot. Car must be taken to include intra-process and inter-process correlations. The number of packets, for example, might correlate with the number of sent bytes.

6. EXAMPLES:

6.1. MODELLING WEB TRAFFIC

In this scenario, the user models will only simulate web user, i.e. users surveying through the WWW by using web browsers. In principle, such users will be able to open web documents with other applications as well (for example by klicking on a document in the web browser and then choosing *Open Document,* the web browser will download the document and then launch the appropriate application associated with this Mime type automatically). As the main goal of BISANTE is the simulation of network load, this will be reflected only by the size of the downloaded document and the inter-arrival time between two user actions in the web browser.

6.1.1. WORKLOAD GENERATOR

The workload generator associated with each node set will consist of a simple point process (see Deliverable 1.1) creating user sessions at a pre-defined rate. This rate may also vary as time goes by. As the arrival of users in general will be un-correlated, such a point process is given for example by a Poisson process with constant or variable arrival rate.

6.1.2. SESSION LAYER

As the only application used is a web browser, a dummy model will be inserted that just starts a web browser once and then waits for the end-signal of this application. After receiving the end-signal, the dummy model will send an end-signal to the next higher layer.

6.1.3. APPLICATION LAYER

At this layer, the next command sequence (and thus the services to use) are chosen. For simplicity reasons, only command sequences starting and stopping HTTP services are considered.

The models thus will consist of a point process yielding the point of time to start the next command sequence, as well the defining the command sequence type.

For example, the application model could choose between starting HTTP GET or HTTP POST. At this layer, there might be correlations between successive HTTP types. This can be reflected by a simple two-state Markov chain, one state starting the HTTP Get command sequence, the other HTTP POST. Additionally, statistical tests should be applied to find out the necessary order of this Markov chain.

6.1.4. COMMAND LAYER

The user command layer again consists of a dummy layer. Upon receiving SIG_START it will start the appropriate HTTP service (GET or POST) and then will wait for its completion). At service completion, it will send SIG_END to the next higher level.

Additionally, the probability for stopping the running HTTP service according to the overall run-time can be modelled as QoS checks. This can be done by a Poisson process, where the arrival of the entity will trigger a QoS dummy action to send SIG_STOP to the service layer.

6.1.5. SERVICE LAYER

The service layer will in principle model a full-duplex communication over one or various TCP connections.

At service start, this model will open a TCP connection to the stated server and will send a HTTP command message to it. After receiving the result frame, the model will decide, how many embedded objects to download further (in case of HTTP Get). Depending on the HTTP version, the model will then download the remaining objects over the same or newly created TCP connections.

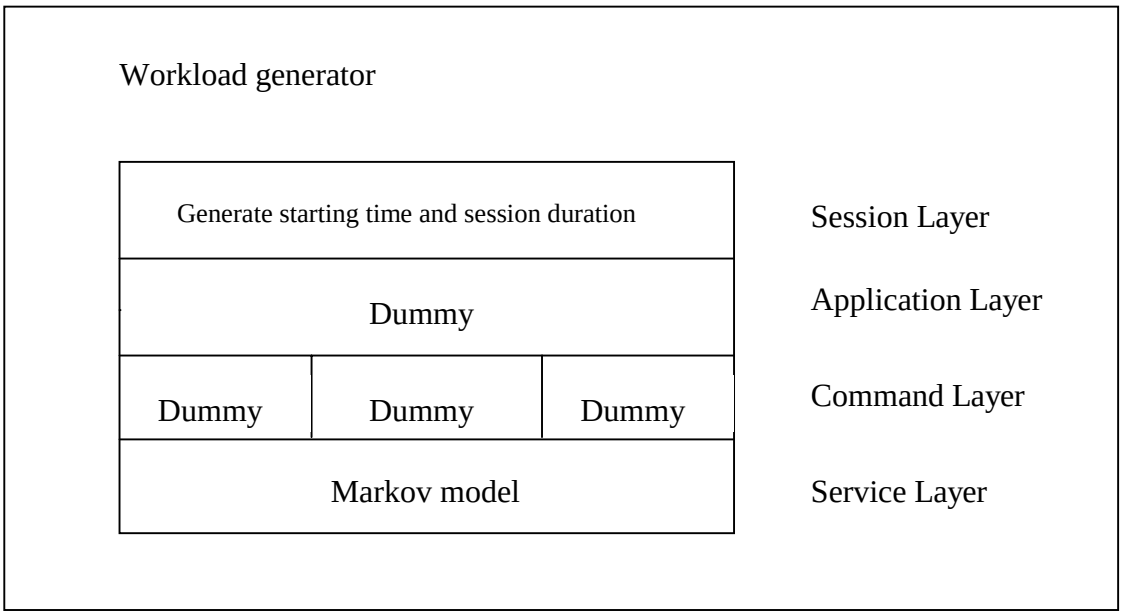
If all embedded objects have been downloaded successfully, the model will send a SIG_RECV and SIG_END to the next higher layer.

The file sizes of the downloaded documents can either be created in this layer, or at the service layer of the requested web server.

User profiles of this structure will additionally contain values for the model parameters, such as observed Poisson arrival rates, the Markov chain order and transition probabilities etc.

6.2. MODELLING ATM CELL FLOW FOR MBONE TRAFFIC

In this section, a model of ATM cell flow for an MBONE traffic will be given (cf. D1.1 A.2). As we do not have data about the way in which the sessions are generated, we can only analyse the traffic and model it in order to obtain a traffic generator.



6.2.1. WORKLOAD GENERATOR

The workload generator consists of a dummy layer, which only creates a user session. When the workload generator is run, SIG_INIT and SIG_START signals are sent to the session layer.

6.2.2. SESSION LAYER

The duration of the session and the starting time can be chosen at the session level, according to a probabilistic process or a deterministic one. This layer starts and stops the session by sending SIG_INIT and SIG_START signals to the lower layer. When the end of the session occurs, SIG_END and SIG_STOPPED signals are sent by the session layer.

6.2.3. APPLICATION AND COMMAND LAYERS

Once a broadcasting session has been started, there is no further user interaction at the application and/or command layers. Therefore, the application and command layers are modelled as dummy layers, which only transmit the start/stop signals to the service layer.

6.2.4. SERVICE LAYER

The service layer models the traffic generator in terms of the flow of ATM cells. When this service starts, an ATM connection is opened and traffic is generated according to a model, which was chosen to represent the nature of traffic (burstiness, long range dependence) as observed from measurements (see Deliverable D1.2-Case Studies for further details).

7. SUMMARY OF THE RESULTS

Based on the layered framework identified in D1.1 of the project, a knowledge based approach for user behaviour modelling to be used in network traffic simulation was proposed.

To sumarise, the advantages of the layered modelling approach are outlined as follows:

- **Ease of Use**: A user behaviour profile is a self-contained description of the models and parameters representing a certain class of users in a particular environment. Such a profile is ready-to-be-used as a workload generator feeding a network simulation.
- **Flexibility**: The layered framework allows the analyst to construct the model at the desired level of detail. He or she could start for example with an existing profile, adding details where necessary and/or omitting layers which are not relevant in his or her particular sutdy.
- **Re-usability**: As interfaces between layers are well defined, models can be re-used.
- **Aggregation** techniques can be applied to simplify model construction and evaluation.
- The **interface** to the network simulation components is well defined at the lowest layer of the hierarchy, thus decoupling the task of workload modelling from the task of modelling the network part as much as possible.

Future work will focus on the elaboration of the case studies. The first two case studies will mainly serve to demonstrate the approach, while the latter two will be used for validation.

In the next deliverable, D1.3, a detailed description of the type of models to be applied at each layer will be given along with a motivation, why those models have been chosen. The user profiles for the case studies will also be presented.