Lightweight Testing of Communication Networks with *e-Motions*

Javier Troya, José M. Bautista, Fernando López-Romero,
and Antonio Vallecillo

GISUM/Atenea Research Group. Universidad de Málaga, Spain
{javiertc,jmbautista,fernando,av}@lcc.uma.es

Abstract. This paper illustrates the use of high-level domain specific models to specify and test some performance properties of complex systems, in particular Communication Networks, using a light-weight approach. By following a Model-Driven Engineering (MDE) approach, we show the benefits of constructing very abstract models of the systems under test, which can then be easily prototyped and analysed to explore their properties. For this purpose we use *e-Motions*, a language and its supporting toolkit that allows end-user modelling of real-time systems and their analysis in a graphical manner.

1 Introduction

Lightweight modelling is the use of small, abstract models of the system under study, and of push-button verification techniques [1]. The key ideas behind this approach, as proposed by Pamela Zave, are the construction of a very abstract model of the system and the use of analysis tools to explore its properties. Being the model very abstract in comparison to a real implementation, and focusing only on the relevant concepts, it becomes small, tractable, and can be constructed quickly. Being the analysis tools simple and pushbutton-based, they yield results with little effort. Thus it becomes easy for the system designer to prototype the system, test its properties and re-adjust the designs in a cost-effective manner. Moreover, this enables an incremental and iterative approach to system design and testing, where the system is progressively specified and its properties analysed for correctness and against a set of quality requirements. The problems found during the testing process can be carefully analysed and either the system design or the quality requirements refined accordingly.

In this paper we show how Domain Specific Modelling Languages (DSMLs) can help realizing this approach. In the first place, they allow end-users to create models of their systems at the right level of abstraction and with the appropriate precision. Secondly, the produced models can be connected to powerful simulation and analysis tools using model transformations, to provide the push-button capabilities required for accomplishing the analysis.

We illustrate our approach using the *e-Motions* language and supporting toolkit [2], which enables the precise definition of real-time models in a graphical and intuitive way, as well as its simulation and analysis [3].
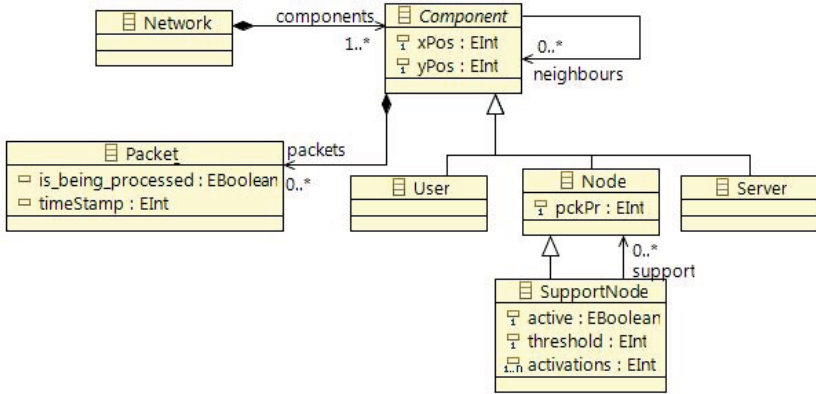
Fig. 1. Communication Network Metamodel

As a running example we use a re-configurable Communication Network system, composed of computers that transmit messages through nodes that process and forward them to other nodes until messages reach their final destinations. Additional supporting nodes can be activated in case of network congestion to alleviate the temporary traffic bottlenecks. Assuming that the cost of acquiring and maintaining these extra nodes is not negligible, there are some tradeoffs between the quality of service provided by the network and its overall cost. We show how this kind of analysis and tests can be conducted with our proposal in an easy and cost-effective manner.

The structure of this paper is as follows. Section 2 introduces the running example and provides the motivation of our work. Then, Section 3 describes the structural model of the system, and them how to model its behaviour so that it can be later simulated and analysed, as discussed in Section 4. Finally, Section 5 describes some related works and Section 6 presents the conclusions.

2 A Running Example

Let us start by describing the system that we want to model and whose performance and behaviour we want to analyse. It consists of a communication network composed of different kinds of Components that can contain Packets. Each component has a specific location, given by two coordinates. The metamodel of the network is shown in Fig. 1.

Users produce packets, while Servers consume them (i.e., they act as sources and sinks of the network, respectively). Components can exchange Packets only if they are connected. Such connection between components is modelled by the neighbours reference, that reflects the components that are reachable from a given component. The network itself is modelled by a set of packet switching Nodes, which are the network elements in charge of receiving, processing and forwarding packets to other components. Nodes have one attribute (pkPr) in our

model to keep track of the packets they have processed so far. The buffer with the set of received packets that a node has to process is modelled by means of the composition relation between Component and Packet.

One characteristic of our network is that the time each node spends in processing a packet depends on the number of packets in its buffer. The more packets in the buffer, the more the node takes to process each one. This simulates a behaviour where nodes need to perform some operations on the flow of packets, such as sorting or merging them according to a given algorithm, for instance. Packets have two attributes, is_being_processed and timeStamp. The former indicates whether the packet is currently being processed by a node, while the latter stores the moment in time at which the packet enters the network. For routing packets, nodes decide to forward packets to the neighbour node which is less loaded, i.e., the one with the smallest buffer size.

In order to alleviate network congestion, an additional kind of nodes (called SupportNodes) exists in the network. They can be activated and de-activated depending on the load of the neighbouring nodes. Each SupportNode activates itself if the number of packets in the buffer of any of the nodes connected to it via the support relationship goes above the value defined in its threshold attribute. Similarly, it deactivates itself when the load of all connected nodes is above the threshold. Attribute activations keeps track of the moments in time at which the support node changes its state.

Let us assume that the cost of acquiring, maintaining and running these extra support nodes cannot be ignored, as it happens for instance if support nodes are hired from external network providers, and their running costs depend on the time they are active or on the number of packets they process. In this setting the system owner is faced with several decisions in order to maximize the quality of service provided by the network while minimizing its overall cost. Firstly, how many supporting nodes need to be hired/purchased to guarantee a minimum level of throughput? Secondly, which is the optimal value for the threshold of each support node that provides a required level of throughput with the minimum time of support node activation (hence minimizing the running cost of the node)?

In order to be able to respond to these questions, we need to identify which are the system parameters that are relevant to our analysis. In our case, we will focus on the following ones:

- **Throughput** and **delay** of the overall network. They indicate how fast nodes process packets. Throughput tells us how many packets are processed by the network per unit of time. Delay indicates how many time units the packets spend within the network. The higher the throughput, the lower the delay, and so the higher the performance of the network.
- **Packets processed per node.** This measure provides an indication of the work load supported by each node. This is however a complex indicator due to the way in which packets are processed in this network, and how they arrive to nodes. The fact that processing time depends on the length of the buffer of pending packets may cause different behaviour depending on whether packets are coming in bursts or at a regular pace.
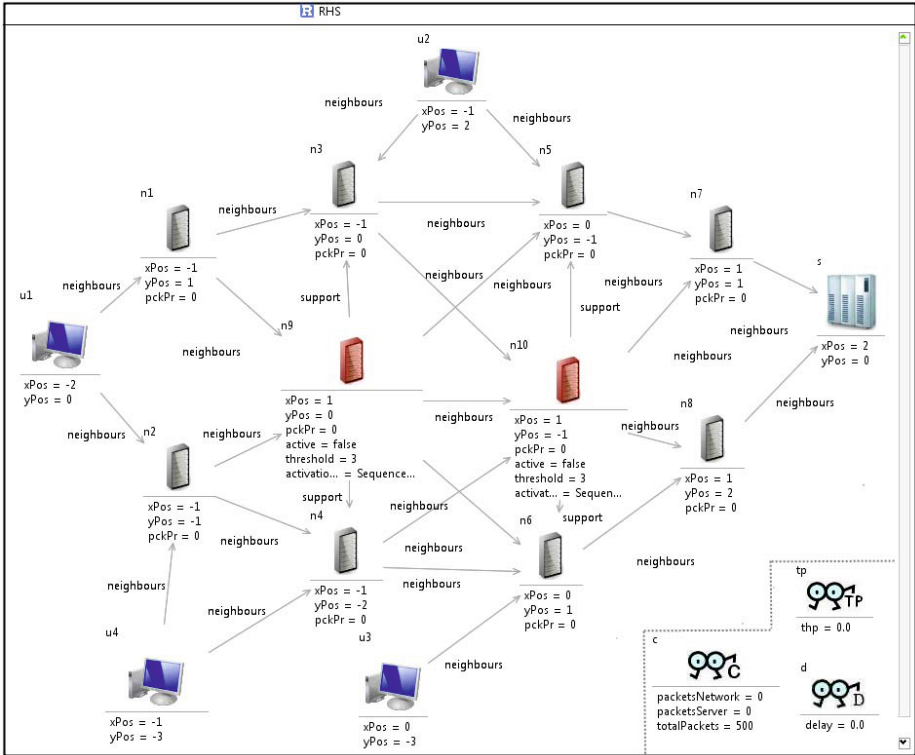
Fig. 2. Initial model of the network

– **Packets processed per SupportNode.** This measure is important because
it provides an indication on the real need of these nodes.
– **Activation times of SupportNodes.** The time and frequency of activation
of this kind of nodes also provides useful information about their actual
usage in the current network configuration.

3 Modelling the Communication Network Using *e-Motions*

3.1 Modelling the Structure

The first step is to model the initial configuration of the system. This is nothing
but a model that conforms to the Network metamodel. A possible configuration
example of a network is shown in Fig. 2 (please ignore the area within the dotted
lines for now). This configuration defines three Users feeding packets into the
network and one Server consuming them. Each user accesses the network using
different nodes. The network is composed of 8 (normal) nodes and 2 support
nodes (n9 and n10), which are initially deactivated. The activation of the support

nodes depends on the buffer size of nodes n3 and n4 for support node n9, and of nodes n5 and n6 for support node n10. This is specified by the corresponding support relations between the support nodes and the nodes they try to help.

3.2 Modelling Behaviour

Apart from the structure of our system, which is captured by the model shown in Fig. 2, we need to be able to describe its behavioural dynamics in a way that allow us to reason about them. One way to do this is by describing the evolution of the modelled artifacts along some time model. In MDE, this can be done using model transformations supporting in-place updates [4]. The behaviour of the system is then specified in terms of the permitted actions, which are in turn modelled by the model transformation rules.

In-place transformations are composed of a set of rules, each of which represents a possible *action* of the system. These rules are of the form $l : [NAC]^* \times$ LHS $\rightarrow$ RHS, where l is the rule's label (its name), and LHS (left-hand side), RHS (right-hand side) and NAC (negative application conditions) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the preconditions for the rule to be applied, whereas the RHS represents its postcondition, i.e., the effect of the corresponding action. Thus, a rule can be applied, i.e., triggered, if an occurrence (or match) of the LHS is found in the model and none of its NAC patterns occurs. Generally, if several matches are found, one of them is non-deterministically selected and applied, producing a new model where the match is substituted by the appropriate instantiation of its RHS pattern (the rule's *realization*). The model transformation proceeds by applying the rules in a non-deterministic order, until none is applicable — although this behaviour can be usually modified by some execution control mechanism, e.g., strategies [5].

3.3 *e-Motions*

In [2] we presented *e-Motions*, a tool for the formal and precise definition of real-time DSMLs in a graphical and intuitive way developed for Eclipse. It extends in-place model transformation with a model of time and mechanisms to state action properties, designed for the specification of Domain Specific Visual Languages (DSVL) of real-time systems. Time-related attributes can be added to in-place rules to represent features like duration, periodicity, etc. Two types of rules were defined to specify time-dependent behaviour, namely, *atomic* and *ongoing* rules. Atomic rules represent atomic actions with a specific duration, which is specified by an interval of time with any OCL [6] expression. In fact, *e-Motions* has full support for OCL thanks to mOdCL [7], which implements and give semantics to OCL in Maude [5]. The mentioned rules can be periodic, i.e., they admit a parameter that specifies the amount of time after which the action is periodically triggered (if the rule's precondition holds, of course). In our latest version of *e-Motions*, probability distributions can be used for specifying these times. In this way, we can, for example, let the arrival of packets to a system depend on a poisson distribution. *Ongoing* rules represent actions that progress continuously
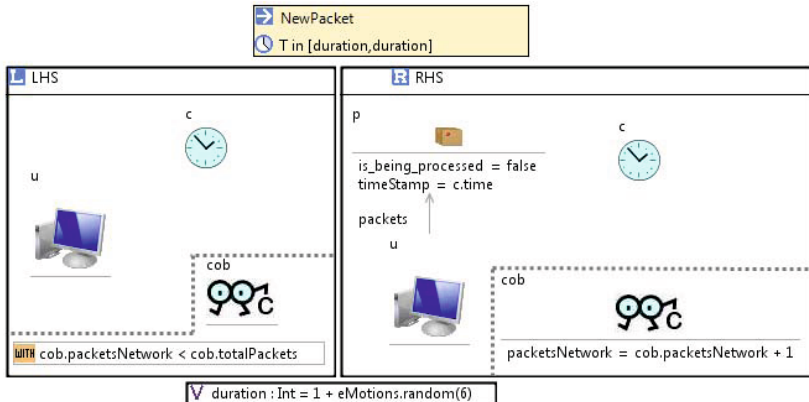
Fig. 3. NewPacket Rule

with time while the rule preconditions (LHS and not NACs) hold. Both atomic
and ongoing rules can be scheduled, or be given an execution interval.

In order to be able to model both state-based and action-based properties, we
have also proposed extending model patterns with action executions to specify
action occurrences. These action executions specify the type of the action (i.e.,
the name of the atomic rule), its status (e.g., if the action is unfinished or real-
ized) and its identifier. They may also specify its starting, ending and execution
time and the set of participants involved in it. This provides a very useful mech-
anism when we want to check whether an object is participating in an action or
not, or if an action has already been executed.

A special kind of object, named Clock, represents the current global time
elapse. Designers are allowed to use it in their timed rules (using its attribute
time) to know the amount of time that the system has been working.

e-Motions offers automated bridges to the Maude [5] executable language
and its formal toolkit. Maude is used as a formal notation to provide the precise
semantics of the corresponding *e-Motions* specifications (as described in [8]),
while at the same time the model transformations between *e-Motions* and Maude
(implemented in ATL [9]) allow the Maude tools to become available in the *e-
Motions* environment. In this way, both simulation and the use of some formal
analysis tools are possible for *e-Motions* specifications [10].

3.4 Specifying the Behaviour of the Network

The behaviour of the network will be specified by a set of rules, each one de-
scribing one possible action.

The NewPacket rule, shown in Fig. 3, simulates the generation of packets
by users. This process follows a uniform distribution in the interval [1,7], i.e.,
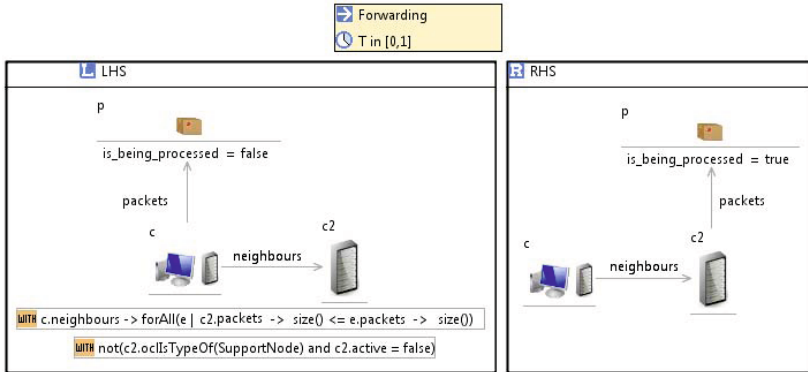a user generates a packet every duration time units. Here, duration determines

Fig. 4. Forwarding Rule

the duration of the rule and is calculated using a random number generator (**eMotions.random(6)** returns a value between 0 and 6). Packet attributes are initialized at creation as shown in the right hand side of the rule.

The Forwarding rule (Fig. 4) models the forwarding of packets among components and nodes. This rule is fired when sending packets from users to nodes and from nodes to nodes. To apply this rule, the packet must not be being processed. Furthermore, there are two OCL expressions that have to be satisfied in order to launch the rule. They state that the target node is the component's neighbour which is processing a lower number of packets, and that it cannot be a deactivated support node. In the RHS pattern of the rule the packet has moved to the node and it has started being processed. The duration of this rule can be either 0 or 1 time units (the fact that a packet can take 0 units simulates the situation in which several packets are forwarded together to optimize an open connection).

In Fig. 5(a) we can see the PacketProcessing rule. It models the processing of a packet by a node by modifying its is_being_processed attribute. The pckPr attribute of the node is increased in one unit as it has processed a new packet. The time this rule spends is directly proportional to the number of packets being processed by the node. PacketArrival rule (Fig. 5(b)) models the arrival and consumption of a packet from a node to the server. The time this rule consumes is either 0 or 1 time units.

Finally, activation and deactivation of support nodes is specified by two rules. ActivationSupport rule (Fig. 6(a)) deals with the activation of a support node when it is deactivated and one of the nodes it supports is processing more packets than indicated by the node threshold. DeactivationSupport rule (Fig. 6(b)) carries out the opposite action. In both rules, the time unit when the activation/deactivation occurs is added to the node's activations attribute. These rules are instantaneous rules, i.e., atomic rules with duration 0.
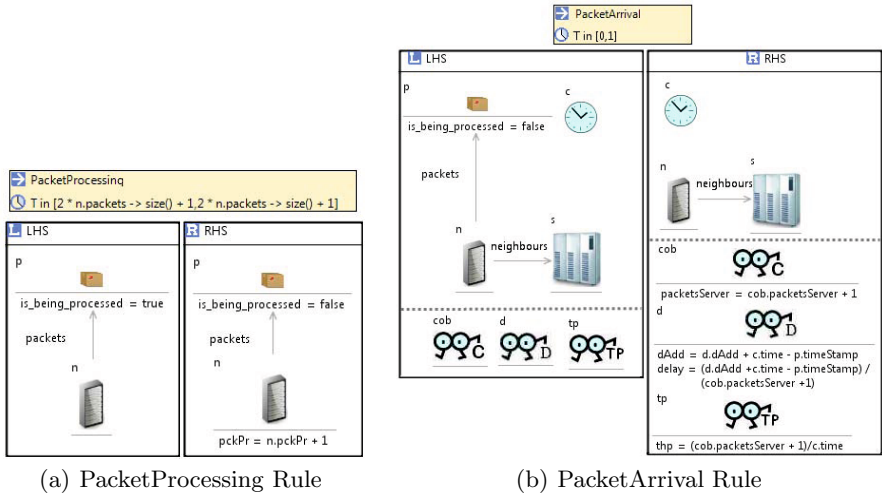
(a) PacketProcessing Rule

(b) PacketArrival Rule

Fig. 5. PacketProcessing and PacketArrival Rules

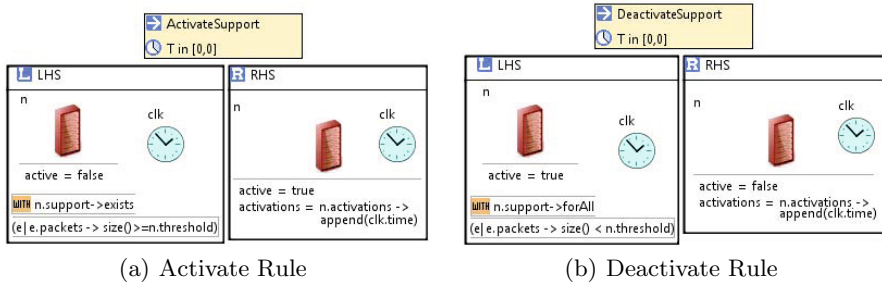(a) Activate Rule

(b) Deactivate Rule

Fig. 6. Activate and Deactivate Rules

3.5 Adding Observers for System Monitoring

Apart from the intrinsic properties of the system, there are also other features that we may need to express and capture in our models. For example, in this network we are interested in monitoring the throughput and delay of the packets processed by the network as well as in the number of packets processed by each node, especially the support nodes. The activation/deactivation frequency of the support nodes is also relevant. Although some of these properties could be analysed using the model element attributes (e.g., number of processed packets), other features should be expressed using additional elements.

The traditional solution has normally consisted in extending the system meta-model with additional attributes, i.e., extending the structure of the system to accommodate new state variables. In [3] we introduced *observers* for tackling this problem using a modular and reusable approach.
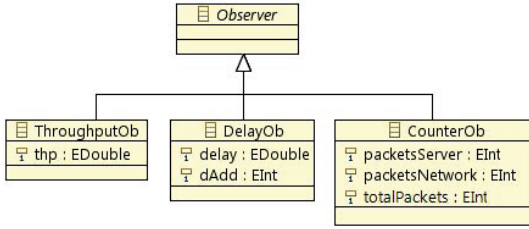
Fig. 7. Observers Metamodel

An observer is an object whose purpose is to monitor the state of the system: the state of the objects, of the actions, or both. Observers, as any other objects, have a state and a well-defined behaviour. The attributes of the observers capture their state and are used to store the variables that we want to monitor. We have defined an *Observers metamodel*, which is shown in Fig. 7. We have three different observers:

- ThroughputOb, in charge of monitoring the throughput of the system (the number of packets processed by the network per time unit).
- DelayOb, that tracks with its delay attribute the average time spent by packets to be processed by the network.
- CounterOb, responsible for counting packets. The packetsServer attribute counts the number of packets that arrive at the server. It is used to calculate the throughput and delay of the system. Attribute packetsNetwork stores the number of packets that users introduce in the network. Finally, totalPackets determines an upper limit for the simulation process, specifying the total number of packets that the network will process.

The idea for analysing the system with observers is to combine the original metamodel (Fig. 1) with the Observers' metamodel (Fig. 7) to be able to use the observers in our DSVL. Since *e-Motions* allows users to merge several metamodels in the definition of the behaviour of a DSVL, we can define the *Observers metamodel* in a non-intrusive way, i.e., we do not need to modify the system metamodel to add attributes to their objects. Furthermore, this approach also enables the reuse of observers across different DSVLs. The behaviour of the observers is specified using rules, too.

To specify how observers monitor the non-functional properties of the system we have included them in the rules (inside the area delimited by dotted lines—these dotted lines do not form part of the rules, they have been added to the diagrams of this paper for understandability reasons). Thus, starting with the initial model of the system (Fig. 2), we see how we include an observer of each type in the network and we give their attributes some initial values. We see that the network will process up to 500 packets. Continuing with the New-Packet rule (Fig. 3), we use here the CounterOb observer to stop users generate packets when the specified upper limit is reached. PacketArrival rule (Fig. 5(b))

models the arrival of a packet to the server, updating the three observers' state appropriately: the CounterOb observer updates the number of packets arrived to the server; the DelayOb observer updates its attributes to properly compute the delay, and finally the ThroughputOb computes the current throughput.

4   Simulating and Analysing the Network

Once the specifications are written, this section describes how they can be simulated and analysed with the *e-Motions* tool.

In *e-Motions*, the semantics of the real-time specifications is defined by means of transformations to other domain with well-defined semantics, namely Real-Time Maude [11]. The *e-Motions* environment not only provides an editor for writing the visual specifications, but also implements their automatic transformation (using ATL) into the corresponding formal specifications in Maude.

One of the benefits of this approach is that it allows to make use of the Maude facilities and tools available for executing and analysing the system specifications once they are expressed in Maude. In [10,8] we showed some examples of analyses that can be performed on the Maude specifications. Furthermore, Maude rewriting logic specifications are executable, and therefore they can be used as a prototype of the system and to run simulations.

In Maude, the result of a simulation is the final configuration of objects reached after completing the rewriting steps, which is nothing but a model. This resulting model can then be transformed back into its corresponding EMF notation, allowing the end-user to manipulate it from the Eclipse platform. The semantic mapping as well as the transformation process back and forth between the *e-Motions* and Real-Time Maude specifications is described in detail in [8], although it is completely transparent to the *e-Motions* user. In this way the user perceives himself as working only within the *e-Motions* visual environment, without the need to understand any other formalism or being completely unaware of the Maude rewriting engine performing the simulation.

Regarding the use of the resulting models by other tools, *e-Motions* implements a trivial model-to-text transformation that enables the creation of a *.csv* file from an Ecore model. Such a *csv* file contains the information of every object in the model, together with the values of all its attributes. Objects are named by their identifiers, and attributes are expressed as a list of name-value pairs. Such file can be directly imported by different applications for performing different kinds of analysis. For example, it can be fed to an spreadsheet application that the domain expert can use to analyse the data, display charts, etc. In this way, the domain expert will be able to easily display charts with the result of a simulation (which is in fact a model) to graphically represent the values of the parameters monitored by the observers throughout the whole simulation.

4.1   Tests

In order to understand how the network works and in order to analyse the parameters mentioned in Section 2, we have simulated the network with different
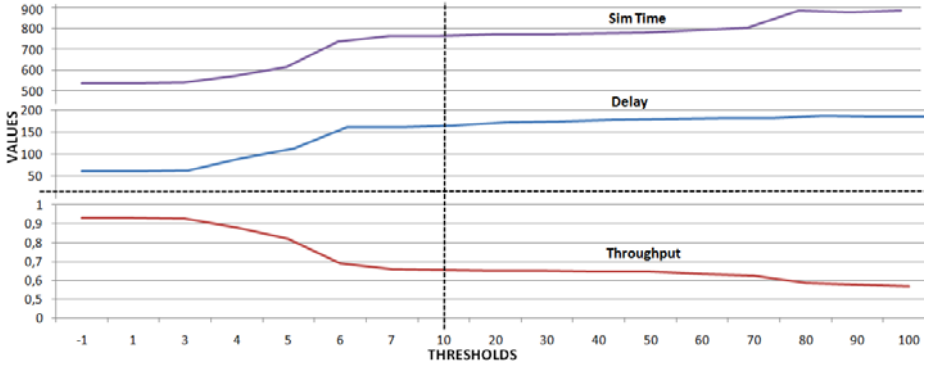
Fig. 8. Simulation time, delay and throughput

threshold values for the support nodes. They have ranged from −1 (the support node is always active) to 100 (they will never be active because the buffers of the nodes in our example keep always below that value). For every threshold value we have run five different simulations since users introduce packets in the network in a random manner. The figures showed in the charts correspond to the average results for the obtained values. In all the simulations, we have limited the number of packets that enter the network and reach the server to 500.

Fig. 8 shows the values of throughput, delay and the time units taken by the simulations. Most variations occur when the threshold is between −1 and 7, before they become stable. This is why the chart is divided in two horizontal parts, in which the left part zooms out the $[1, 10]$ interval. The vertical axis has also been split into two sections, in order to distinguish the area where the throughput values reside.

The examination of the chart reveals that, as expected, the best performance (highest throughput, lowest delay and lowest simulation time) is achieved when the support nodes are always active (threshold $= -1$). However, this is also the most expensive situation. The behaviour of the support nodes turns out to be more interesting when the thresholds are between 3 and 6. In that range, the three parameters experiment the biggest variation, making the network slower as the thresholds increase. We can also see in the right part of the chart a variation in the simulation time and throughput that is a bit more pronounced than the rest. It is between thresholds 70 and 80. We give an explanation to this fact when we discuss the chart shown in Fig. 9.

Fig. 9 shows the chart with the number of packets processed by the two support nodes for each threshold value. We see that the second support node (node n10 in Fig. 2) processes packets when its threshold is within the range $[-1, 7)$. In fact, in the range $[-1, 5]$ it processes more packets than the first support node (node n9 in Fig. 2). However, this latter node keeps on processing packets until the threshold is 80. These results were initially unexpected, and
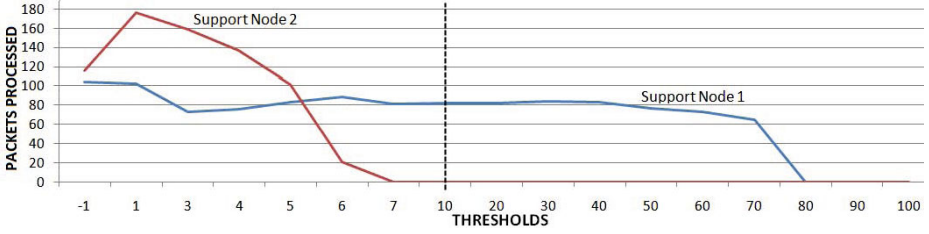
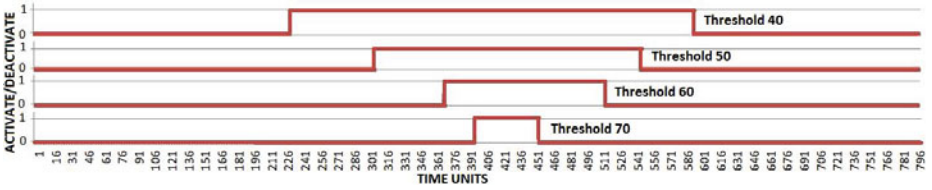**Fig. 9.** Packets processed by the support nodes 1 (n9) and 2 (n10)

Fig. 10. Activation/Deactivation of support node 1 (n9)

they are a result of the topology of the network and the way in which the packets are processed. By defining a different topology to the network and/or by changing the algorithm used to process packets to speed it up (i.e., changing the duration of rule PacketProcessing), the results would be different. Regarding the the behaviour of the system in the range [70, 80], the slope is more pronounced there because this is the threshold value from which the first support node (n9) stops processing packets, i.e., it is not needed at all.

Focusing on the first support node (n9), it finally stops processing packets when the threshold value is 80. By looking at its behaviour in the range [−1, 7], we cannot expect when this node will stop processing packets. In fact, looking at the behaviour of the second support node (at the beginning it processes more packets than the first support node but then it stops processing packets from the threshold value 7), we may expect that the first support node will stop processing packets earlier than it actually does. To help us see how the activation/deactivation of a support node evolves, we have also displayed charts for it. Thus, Fig. 10 shows four charts with the activation/deactivation of the first support node when the threshold values are 40, 50, 60 and 70. We see how the activation of the node is carried out later when the threshold increases. This is because, as thresholds increase, the nodes being supported do not need the help of the support nodes very soon. Support nodes are also deactivated earlier when the threshold is smaller. We have to clarify here that the fact that a support node is deactivated does not mean that it stops processing packets. In fact, it only means that new packets stop coming in, but the node still has to process its buffer of pending packets.
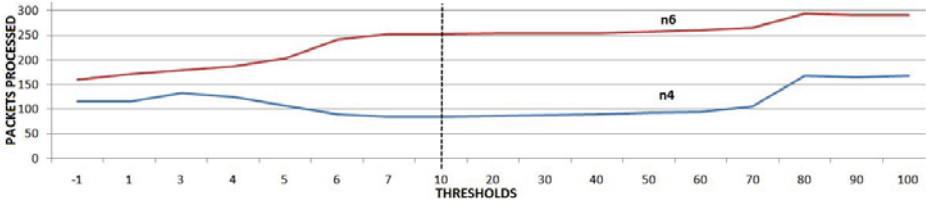
Fig. 11. Packets processed by nodes n4 and n6

Not all the graphs of activations/deactivations are as uniform as the four shown here. The complete set of charts and values obtained for the simulations can be consulted in [12].

Finally, let us show a graph that we also consider of interest (Fig. 11). It displays the number of packets processed by two nodes, n4 and n6 in Fig. 2). This graph is related to the one shown in Fig. 9, since the processing of packets by the support nodes makes nodes n4 and n6 process less packets. In general, we can see that the more packets the first support node (n9) processes, the less packets n4 processes, and the same thing happens with the second support node and n6. For every threshold value, n6 processes more packets than n4 because one of the users sends packets directly to n6.

5 Related Work

There are many different proposals for monitoring and improving the performance and reliability of communication networks, from different perspectives.

In the first place we have those approaches that focus on the actual systems and not on their prototypical models, such as [13,14,15,16]. These works measure the performance of existing network connections (ATMs, multimedia networks, etc.) using dedicated tools. Of course their accuracy and level of precision is very good, but they cannot be used in a predictive way. In other words, these methods and tools are excellent for *a posteriori* testing the network and for checking that it behaves as expected, but they cannot be used for planning purposes in the very early phases of the network design to, e.g., evaluate design alternatives or different routing protocols.

Other kind of approaches focus on design models of the system, before it is actually built. In these (model-driven) approaches, a prototype model of the system is constructed prior to the actual development and deployment, and then analysed for performance or reliability [17,18]. Model-driven proposals can be differentiated depending on three main characteristics: the level of detail used in the models (from very abstract to very detailed); the kind of analysis that they allow (analytical methods, such as Queueing Networks; formal analysis based on the exhaustive exploration of the execution tree, such as model checking; or analysis methods based on simulation techniques); and the level of flexibility provided by the supporting tools.

Some proposals, such as [18,19,20], are based on UML for modelling systems and networks, and normally make use of UML profiles like MARTE [21] for annotating the models with the specification of QoS and other quality properties. These approaches normally provide considerable level of detail and tend to be very precise. Moreover, their models can be transformed into other formalisms such as Stochastic Petri Nets (SPN) [22], Queue Network Models (QNM) [23] and Stochastic Process Algebras (SPA) [24] for performance or reliability analysis. As weak points, their specifications normally remain at a low level of abstraction, and require skilled levels of expertise from the user. Furthermore, their corresponding analytical models can handle certain types of behaviours, but they are limited when the behaviour of the system does not follow specific patterns. For example, modelling networks with complex forwarding algorithms or packet arrival times that do not follow negative exponential distributions are hard to model and to analyse with these approaches. This is something at which we are very strong with *e-Motions*, being able to simulate models whose behaviours follow different distributions [3].

Flexibility is another essential characteristic of any modelling approach in order to be useful during the first phases of the design, but it is a common limitation of many existing approaches. For example, [25] presents a powerful proposal for evaluating the performance of packet switching communication networks using stochastic processes. However, they have a fixed routing strategy and the way of specifying the network is also fixed. With our approach, many different types of networks can be modelled (by simply changing the metamodel or the rules) and different properties can be easily observed.

Visual languages based on graph transformations seem to provide the level of flexibility required for specifying the structure and behaviour of this kind of systems. For example, Reiko Heckel specifies in [26] two protocols for reconfiguring P2P networks, and analyse their reliability. Protocols are modelled using stochastic graph transformations, which, as in our approach, take advantage of negative application conditions (NACs) and path expressions. For the specification and analysis of the system he uses model checking, chaining several tools (namely *GROOVE* and *PRISM*). This approach allows very interesting and useful kinds of analysis, but it also has some limitations. Firstly, even the very high-level models of the system cause a state explosion that can become unmanageable very soon. Secondly, in this approach users have to change the modelling environment when moving from the system design to the analysis, with the need to be familiar with more than one environment. The first problem is intrinsic to the complexity of the networks to analyse and this is why simulation is sometimes more effective to reveal design problems, specially in the early phases of the system design (at least, until the structure and dynamics of the system become stable). Then model checking or any other tool-supported mechanism that allows exploring the execution tree may be used. In *e-Motions* we can use not only simulation but also Maude's `search` facilities, without having to escape the *e-Motions* environment.

There are also other interesting approaches for modelling and analysing networks at a very high level of abstraction. For example, de Lara et al. present in [27] a DSVL for the definition of traffic networks, using ATOM[3]. Once the network is designed, they map the system models into both untimed and timed Petri nets and show how Petri net analysis and synthesis techniques can be effectively used to analyse the models. In our approach, we map our model into Real Time Maude, and the simulations performed are executed in Maude using its rewrite engine — but in a transparent way to the user. There are many other proposals based on graph transformations that allow modelling timed behaviours [28,29,30,31,32]. However, none of them allows the use of OCL for specifying expressions in attribute calculations and in rule durations. In addition to the flexibility it offers, the expressiveness provided by OCL becomes very useful for specifying complex behaviours at the right level of abstraction and using an appropriate notation.

The final group of works that is related to our proposal allows conducting simulations of the network models. We already mentioned the fact that the analyses based on the exploration of the execution tree may be too heavy-weight for the early phases of system design. And analytical methods, such as QNM or SPA may not be expressive enough to capture some particular characteristics of the systems under study. This is when simulation techniques for analysing performance requirements can be very useful. This is especially important in wireless self-organizing networks (WSONs), which need to be able to respond to dynamic changing environments, operating conditions and practices of use, in a robust way. In fact, the success of WSON-related applications seems to be strongly related to the validation of properties of the network protocols used in these systems. In [33] Alina et al. present a survey with a comprehensive review of the literature on protocol engineering techniques and they discuss the challenges imposed by WSONs to the protocol engineering community. Many of the approaches presented in that paper can be extended to other kinds of networks. They present formal and non-formal approaches. With respect to the latter ones, in many works, e.g., [34,35,36], simulation is used to check protocol designs. They also use the concept of monitors, as we use observers, to define entities that check the performance metrics during the simulation executions and generate the traces files accordingly. However, most of these proposals do not use model-driven techniques: they implement the algorithms in general purpose programming languages such as Java. This results in expensive development costs and efforts, in lack of flexibility and in error-prone simulators due to the complexity of the systems to simulate. Other works (e.g., [37]) have developed prototyping environments for certain kinds of networks, in order to avoid this problem. This is also what we have done, but showing how the use of domain specific languages and a model-driven environment such as the one provided by *e-Motions* can be even more flexible, eliminating the restriction of having to deal with particular kinds of networks (the ones for which the simulation framework was developed).

The basic *e-Motions* language has been presented in other papers, e.g., [2,3,8]. This paper demonstrates how some of its features and mechanisms can be combined and used to accomplish lightweight modelling and testing of systems, and in particular of communication networks.

6 Conclusions and Future Work

In this paper we have presented a lightweight approach for the design and analysis of non-trivial systems, and showed how these can be realized using *e-Motions*. A communication network example has been used as a proof-of-concept of our proposal, although similar kinds of systems and analysis can be applied in other environments such as P2P networks or the Cloud [20].

There are several lines of work that we would like to explore next. For example, we would like to assign probabilities to the rules. In this way, apart from needing a match of the LHS pattern of the rule, we would also need some probability parameter to be satisfied in order to fire the rule. The use of a similar approach to the one used in Probabilistic Rewrite Theories (pMaude) [38] could be interesting, although other possibilities can also be considered. We also plan to connect *e-Motions* to other interesting Maude tools, such as the LTL model checker [5]. The connection is now possible but requires human intervention. Our plans are to fully integrate some of these tools so that they become accessible to the user in a transparent way. Finally, we are defining connections to other formalisms in addition to Maude, in order to make use of the tools available in these semantic domains for performance and reliability analysis. In particular we are considering connections with Stochastic Petri Nets (SPN), Queue Networks Models (QNM) and Stochastic Process Algebras (SPA), and their associated tools. Although the expressiveness of these notations is different from the expressiveness of *e-Motions*, having access from *e-Motions* to their analytic tools can be of great help. Making the connections work in both ways can also be interesting. In this way these notations can have a direct access to the simulation facilities provided by *e-Motions*.

References

1. Zave, P.: Lightweight modeling of network protocols,
   `http://www2.research.att.com/~pamela/model.html`
2. Rivera, J.E., Durán, F., Vallecillo, A.: A graphical approach for modeling time-dependent behavior of DSLs. In: Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2009), pp. 51–55. IEEE Computer Society, Los Alamitos (2009)

3. Troya, J., Rivera, J.E., Vallecillo, A.: Simulating domain specific visual models by observation. In: Proc. of the Symposium on Theory of Modeling and Simulation (DEVS 2010), Orlando, FL, US (April 2010)
4. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOPSLA 2003 Workshop on Generative Techniques in the Context of MDA (2003)
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
6. Object Management Group: Object Constraint Language (OCL) Specification. Version 2.2, OMG Document formal/2010-02-01 (February 2010)
7. Roldán, M., Durán, F.: Representing UML models in mOdCL (2008), http://maude.lcc.uma.es/mOdCL
8. Rivera, J.E., Durán, F., Vallecillo, A.: On the behavioral semantics of real-time domain specific visual languages. In: Ölveczky, P.C. (ed.) WRLA 2010. LNCS, vol. 6381, pp. 174–190. Springer, Heidelberg (2010)
9. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)
10. Rivera, J.E., Vallecillo, A., Durán, F.: Formal specification and analysis of domain specific languages using Maude. Simulation: Transactions of the Society for Modeling and Simulation International 85(11/12), 778–792 (2009)
11. Ölveczky, P., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
12. Atenea: Packet Switching Simulation Results (2011), http://atenea.lcc.uma.es/index.php/Page/Resources/E-motions/PacketSwitchingExample/Results
13. Jain, M., Dovrolis, C.: End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. IEEE/ACM Transactions Networking 11(4), 537–549 (2003)
14. Carter, R.L., Crovella, M.E.: Measuring bottleneck link speed in packet-switched networks. Perform. Eval. 27-28, 297–318 (1996)
15. Lindh, T.: Performance management in switched ATM networks. In: Trigila, S., Mullery, A., Campolargo, M., Vanderstraeten, H., Mampaey, M. (eds.) IS&N 1998. LNCS, vol. 1430, pp. 439–450. Springer, Heidelberg (1998)
16. Pacifici, G., Stadler, R.: Integrating resource control and performance management in multimedia networks. In: Proc. of the IEEE International Conference on Communications, Seattle, WA, vol. 3, pp. 1541–1545 (1995)
17. Balsamo, S., Marco, A.D., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Trans. on Software Engineering 30(5), 295–310 (2004)
18. Cortellessa, V., Di Marco, A., Inverardi, P.: Integrating performance and reliability analysis in a non-functional MDA framework. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 57–71. Springer, Heidelberg (2007)
19. Tawhid, R., Petriu, D.C.: Integrating performance analysis in the model driven development of software product lines. In: Busch, C., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 490–504. Springer, Heidelberg (2008)
20. Li, J., Chinneck, J., Woodside, M., Litoiu, M., Iszlai, G.: Performance model driven QoS guarantees and optimization in clouds. In: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, CLOUD 2009, pp. 15–22. IEEE Computer Society, Vancouver (2009)
21. OMG: UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE). Object Management Group (June 2008), OMG doc. ptc/08-06-08

22. Marsan, A.: Stochastic petri nets: An elementary introduction. In: Rozenberg, G. (ed.) APN 1989. LNCS, vol. 424, pp. 1–29. Springer, London (1990)
23. Denning, P.J., Buzen, J.P.: The operational analysis of queueing network models. ACM Comput. Surv. 10, 225–261 (1978)
24. Clark, A., Gilmore, S., Hillston, J., Tribastone, M.: Stochastic process algebras. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 132–179. Springer, Heidelberg (2007)
25. Yaron, O., Sidi, M.: Performance and stability of communication networks via robust exponential bounds. IEEE/ACM Transactions on Networking 1, 372–385 (1993)
26. Heckel, R.: Stochastic analysis of graph transformation systems: A case study in P2P networks. In: Van Hung, D., Wirsing, M. (eds.) Theoretical Aspects of Computing ICTAC 2005. LNCS, vol. 3722, pp. 53–69. Springer, Heidelberg (2005)
27. de Lara, J., Vangheluwe, H., Mosterman, P.J.: Modelling and analysis of traffic networks based on graph transformation. In: Proceedings of the FORMS/FORMATS 2004 Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, Braunschweig, Germany, pp. 120–127 (2004)
28. Burmester, S., Giese, H., Hirsch, M., Schilling, D., Tichy, M.: The Fujaba real-time tool suite: model-driven development of safety-critical, real-time systems. In: ICSE 2005, pp. 670–671. ACM, NY (2006)
29. Gyapay, S., Heckel, R., Varró, D.: Graph transformation with time: Causality and logical clocks. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 120–134. Springer, Heidelberg (2002)
30. Syriani, E., Vangheluwe, H.: Programmed graph rewriting with time for simulation-based design. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 91–106. Springer, Heidelberg (2008)
31. Boronat, A., Ölveczky, P.C.: Formal real-time model transformations in MO-MENT2. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 29–43. Springer, Heidelberg (2010)
32. de Lara, J., Vangheluwe, H.: Automating the transformation-based analysis of visual languages. Formal Aspects of Computing 22(3-4), 297–326 (2010)
33. Viana, A.C., Maag, S., Zaidi, F.: One step forward: Linking wireless self-organizing network validation techniques with formal testing approaches. ACM Comput. Surv. 43, 7:1–7:36 (2011)
34. Girod, L., Elson, J., Cerpa, A., Stathopoulos, T., Ramanathan, N., Estrin, D.: Em*: a software environment for developing and deploying wireless sensor networks. In: Proceedings of the USENIX General Track (2004)
35. Girod, L., Stathopoulos, T., Ramanathan, N., Elson, J., Osterweil, E., Schoellhammer, T., Estrin, D.: A system for simulation, emulation, and deployment of heterogeneous sensor networks. In: Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems, pp. 201–213. ACM Press, New York (2004)
36. Keshav, S.: Real: A network simulator. Technical report, Berkeley, CA, USA (1988)
37. Ben Abdesslem, F., Iannone, L., Dias de Amorim, M., Obraczka, K., Solis, I., Fdida, S.: A prototyping environment for wireless multihop networks. In: Fdida, S., Sugiura, K. (eds.) AINTEC 2007. LNCS, vol. 4866, pp. 33–47. Springer, Heidelberg (2007)
38. Agha, G., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. Electronic Notes in Theoretical Computer Science 153(2), 213–239 (2006)