Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

SIMULATION VISUALIZATION OF DISTRIBUTED COMMUNICATION SYSTEMS

Mihal Brumbulli
Joachim Fischer

Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, GERMANY

ABSTRACT

Simulation is a popular method used for analysis and validation of distributed communication systems due to their complex dynamics. Visualization has proven to be an added value especially when large networks are concerned. In this paper we propose a novel approach and tool support for simulation visualization of formally described distributed communication systems. The system is modeled using Specification and Description Language Real Time (SDL-RT) and a simulation model for the ns-3 network simulator is automatically generated. Network visualization is used in combination with Message Sequence Charts (MSC) for providing detailed visual insight into system dynamics. System validation is also made possible because of the formal semantics of MSCs.

1 INTRODUCTION

Simulation is a popular method used for analysis and validation of distributed communication systems due to their complex dynamics. Analysis and validation is often based on a set of simulation traces, which are a formatted representation of simulation events. These traces present a considerable amount of detail that can potentially become difficult to comprehend without the support of tools. Visualization tools have proven to be very useful especially when large networks are concerned. They use simulation traces to provide visual insight into system dynamics.

Network simulators like ns-3 (Henderson et al. 2006), ns-2 (Breslau et al. 2000), or GTNetS (Riley 2003) can be used for the simulation of distributed systems. These simulators produce traces which can be visualized using tools like Nam (Estrin et al. 2000), iNSpect (Kurkowski et al. 2005), or NetViz (Belue et al. 2008). Scalability is considered a major advantage of this approach, because it can handle networks with hundreds of nodes. Still, the implementation of a simulation model remains a tedious task, because everything has to be hand-coded using general purpose languages like C/C++. This task can be time consuming and error-prone, the models will soon become difficult to maintain, and they cannot be used within other simulation frameworks, except the one they were implemented for. Visualization, on the other hand, focuses on network dynamics. There exist no means for displaying system dynamics at the node level. The available tools use packet-based visualization, which hides the information on real messages exchanged between the nodes.

But network simulation and visualization is not the only approach. The SDL+ methodology (ITU-T 1997) defines how simulation can be used for the analysis and validation of communication systems. Because of their formal semantics and graphical representation, Message Sequence Charts (MSC) (ITU-T 2011) are used for test case specification and simulation tracing. Doldi (2003) gives an insight on the methodology from a more practical perspective. The approach focuses on detailed simulation and visualization of system dynamics. Having that said, it becomes easy to foresee scalability issues especially when distributed communication comes into play. It is nearly impossible to visualize (in a comprehensible way) the dynamics of systems running on distributed environments with hundreds of nodes.

The solution presented in this paper aims at exploiting the advantages of both approaches. Specification and Description Language Real Time (SDL-RT) (SDL-RT Consortium 2006) is used to describe simulation models for distributed communication systems. Implementation for the ns-3 network simulator can be automatically derived from model descriptions via code generation. Network visualization is used in combination with MSCs for scalability and detailed depiction of system dynamics. For this, two levels of visualization are defined: node and network. MSCs are used at the node level. At the network level we use message-based network visualization with MSC semantics. The use of MSCs ensures visualization with formal semantics at both levels, which provides also means for system validation as described in ITU-T 1997.

Section 2 gives an overview of the modeling approach. It shows how SDL-RT can be used to describe simulation models and configurations. These descriptions are then used as a basis for C++ code generation for the ns-3 network simulator, as shown in Section 3. Our visualization approach and tool support is described in Section 4. Finally, we present the conclusions of our work in Section 5.

2 SPECIFICATION AND DESCRIPTION LANGUAGE - REAL TIME

SDL-RT is based on the SDL standard (ITU-T 2007) extended with real time concepts, which include (SDL-RT Consortium 2006):

- use of C/C++ instead of SDL for data types,
- use of C/C++ as an action language, and
- semaphore support.

These extensions considerably facilitate integration and usage of legacy code and off the shelf libraries such as real-time operating systems, simulation frameworks, and protocol stacks (SDL-RT Consortium 2006). Ahrens et al. (2009) and Blunk et al. (2011) show how SDL-RT and simulation frameworks can be used in the development of complex distributed systems. Fischer et al. (2012) have successfully applied this approach in the development of a wireless mesh sensing network for earthquake early warning.

Further extensions to SDL-RT are provided by UML diagrams (OMG 2011):

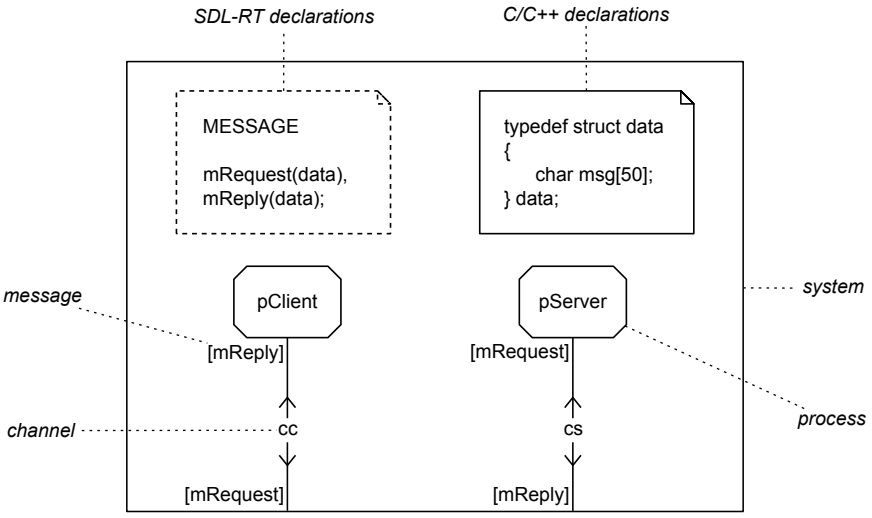
- Class diagrams bring a graphical representation of the classes organization and relations.
- Deployment diagrams offer a graphical representation of the physical architecture and how the different nodes in a distributed system communicate with each other.

SDL-RT can be seen as a pragmatic combination of the standardized languages SDL, UML, and C/C++ for modeling different aspects of real-time systems. These aspects include: architecture and behavior, communication, and deployment. The following paragraphs focus on each of these aspects in terms of a simple client-server application example.

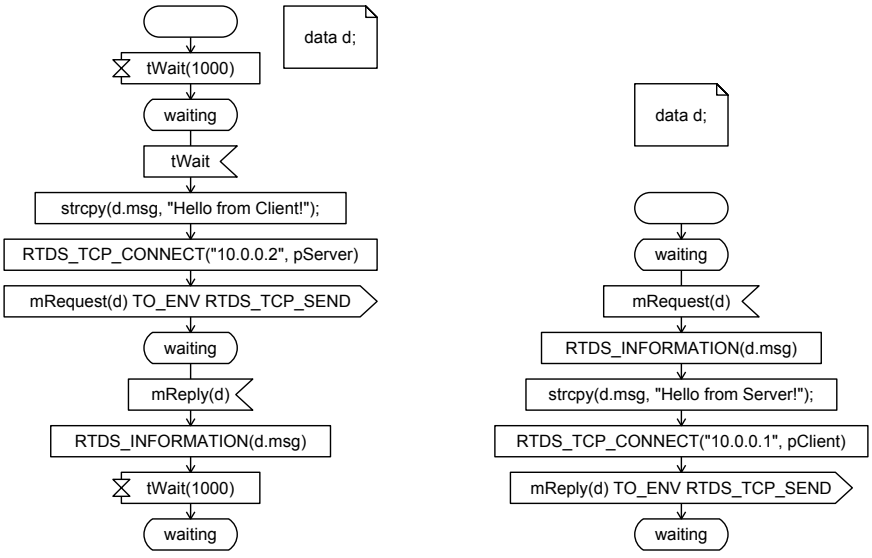
2.1 Architecture and Behavior

In SDL-RT the overall design is called the *system*. It can be composed of *agents* and *communication constructs*. There are two kinds of agents: *blocks* and *processes*. A block is a structuring element that does not imply any physical implementation on the target. Blocks can be composed of other agents and communication constructs. When the system is decomposed down to the simplest block, the way the block fulfills its functionality is described with processes. A process provides this functionality via extended finite state machines. It has an implicit message queue to receive *messages*. A message has a name and a parameter that is basically a pointer to some data. Messages go through *channels* that connect agents and end up in the processes implicit queues. Figure 1a shows these concepts in a simple client-server example.

Everything outside the system is defined as the *environment*. This is considered as a special process and can be used for communication with external entities (i.e. some process that is not part of the system).



(a) Architecture



(b) pClient (c) pServer

Figure 1: Architecture and behavior descriptions for a client-server application in SDL-RT.

In our approach the environment process implements the interface for distributed communication using ns-3 sockets (see Section 2.2). SDL-RT extended final state machines for the client and server processes are shown in Figure 1b and 1c. The descriptions here are quite straightforward:

- The client sends a request message (*mRequest*) to the server and waits for a reply (*mReply*). This sequence of actions is repeated every 1000 ms (tWait timer).
- The server waits for a request from the client. Upon receiving a request, it immediately sends a reply to the client.

2.2 Distributed Communication

The SDL-RT channels cannot describe distributed communication. By definition, they model communication between processes running on the same node.

The simplest solution is to hard code this communication into the description of the process. This is possible because SDL-RT uses C/C++ as an action language. Although it is quite straightforward, this solution has some major drawbacks. First, the models become hard to maintain and debug, and second and most important, coding has to be done for every system description.

Gotzhein and Schaible (1999) introduce a more elegant solution to the problem. They define a set of SDL patterns for the development of distributed systems in Schaible and Gotzhein 2003. We use this pattern-based approach with SDL-RT for modeling distributed communication for simulation purposes. The defined patterns use the ns-3 sockets API for distributed communication via TCP or UDP. Figure 1a shows how this is modeled at the architectural level by simply sending the intended messages to the environment. The pattern is applied as shown in Figures 1b and 1c by using the RTDS_TCP_CONNECT and RTDS_TCP_SEND macros. RTDS_TCP_SEND tells the environment to send the message to the peer process identified by the parameters given to RTDS_TCP_CONNECT.

2.3 Deployment

The SDL-RT deployment diagram describes the physical configuration of run-time processing elements of a distributed system and may contain (SDL-RT Consortium 2006):

- *Nodes* are physical objects that represent processing resources.
- *Components* represent distributable pieces of implementation of a system.
- *Connections* are physical links between nodes or components.
- *Dependencies* from nodes to components mean the components are running on the nodes.

We use the deployment diagram for the configuration of ns-3 simulations. For this purpose we define a set of rules to be applied in diagrams as shown in Figure 2:

- The ≪node≫ type represents a ns-3 node. It has neither properties nor attributes and acts as a container for components. Components represent SDL-RT processes that appear in the system architecture (Section 2.1). Dependencies associate components with ≪node≫ types.
- The ≪device≫ type represents a ns-3 network device. Its type is described as a property of the ≪device≫ (i.e. PointToPointNetDevice in Figure 2). The attributes provide configuration for the specific device. Their names and values can change according to the type of the device.
- The ≪channel≫ type represents a ns-3 communication channel. Its type is also described as a property and the attributes are used for configuration.
- We define two types of connections: ≪node2device≫ and ≪device2channel≫. The first links a ≪node≫ with a ≪device≫; the second links a ≪device≫ with a ≪channel≫.

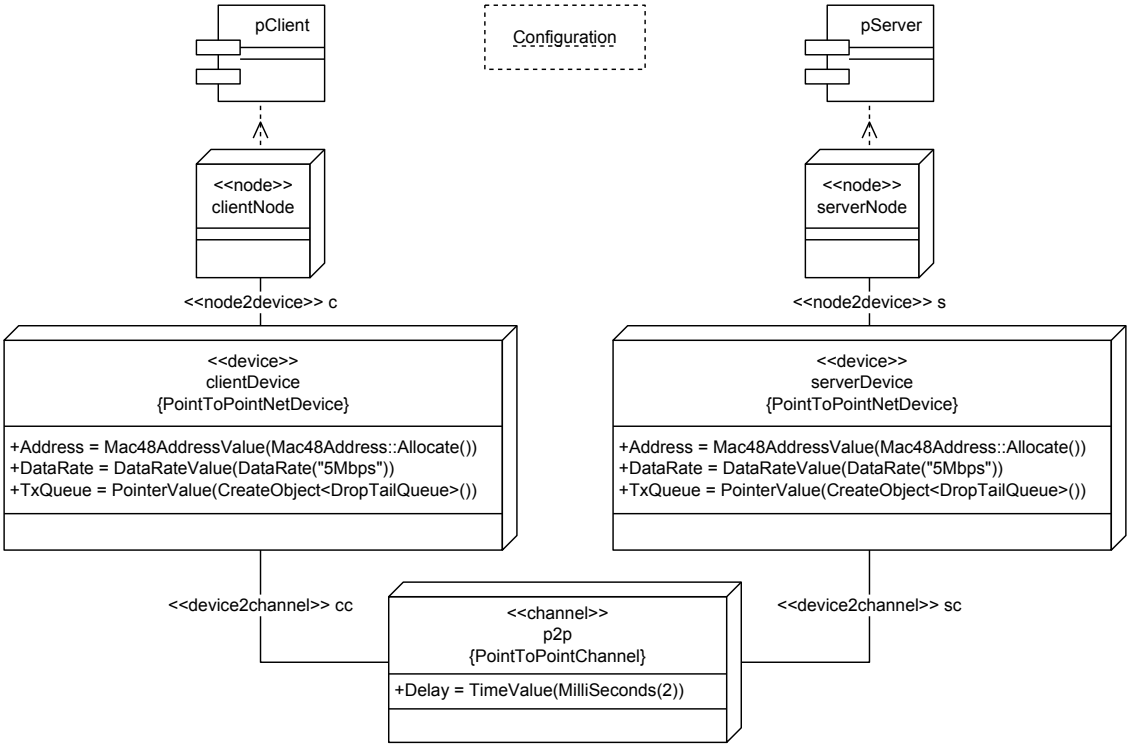


Figure 2: A simulation configuration for the client-server example using SDL-RT deployment diagram.

In addition, ≪node-container≫ and ≪device-container≫ node types can be used for describing configurations with a higher number of nodes. A ≪node-container≫ has only one property, which is the number of nodes to be created. It can be linked with a ≪device-container≫ using a ≪node2device≫ connection.

For more flexibility and support for complex configurations, SDL-RT comments can be used to include C++ code in the description (the *Configuration* rectangle in Figure 2). This code is merged with the generated code as described in Section 3.

3 SIMULATION CODE GENERATION

SDL-RT descriptions (architecture, behavior, and deployment) are used as a basis for the generation of an executable model for the ns-3 simulator. This implies C++ code generation from SDL-RT models.

SDL code generation for network simulators was introduced in (Kuhn et al. 2005). The authors describe how C++ code for the ns-2 simulator can be generated from SDL models. SDL implementations interact with the simulator through named pipes leading to potential scalability issues. We have already addressed this in Brumbulli and Fischer 2011 by generating executables that use directly the simulation library provided by ns-3. We further improve our approach by providing a generic model for code generation as shown in Figure 3.

- *RTDS_InstanceManager* handles the creation of process instances.
- *RTDS_Scheduler* keeps track of all process instances running on a node and handles communication between processes. This communication can be local or distributed. Local communication is handled via shared memory. In this case the sender and receiver process instances are running on the same node, which means that they can be accessed by the same *RTDS_Scheduler*. On the other hand, distributed communication is handled via ns-3 sockets (TCP or UDP), because sender and receiver are running on different nodes. There exists a one-to-one relationship between the

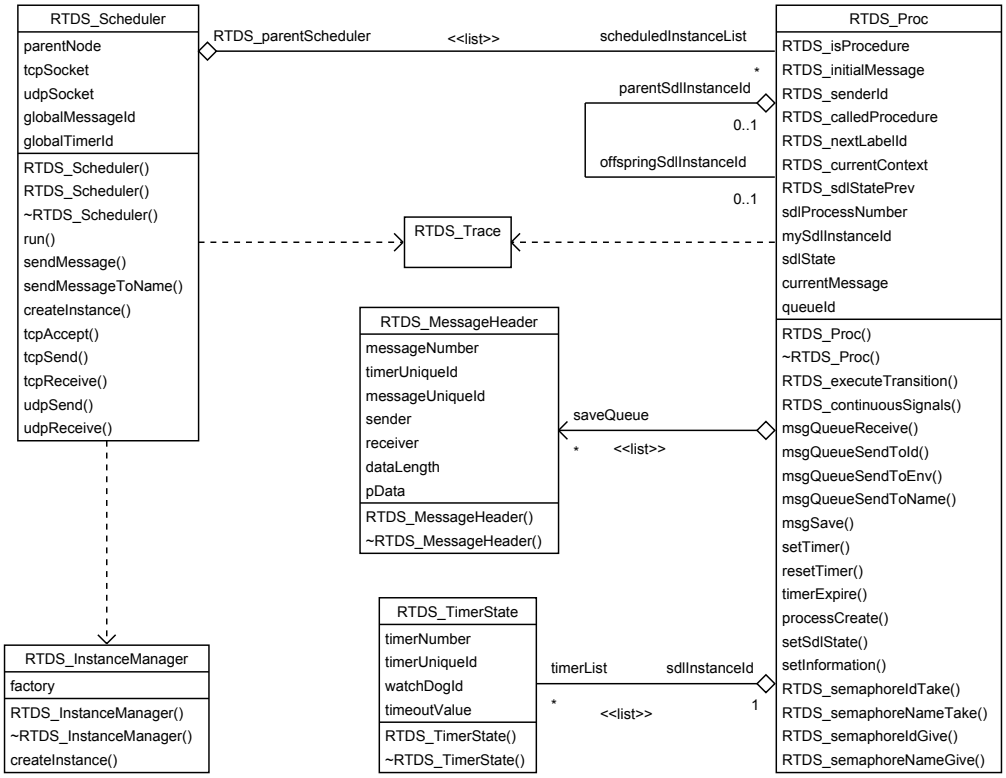


Figure 3: Class diagram for simulation code generation.

RTDS_Scheduler and the ns-3 node (the ≪node≫ type in Figure 2). This concept is implicitly included in the ≪node≫ type definition.

- *RTDS_Proc* provides basic functionality for the SDL-RT processes. All SDL-RT processes (i.e. pClient and pServer in Figure 1) extend this class by implementing the *RTDS_executeTransition* member function. This is the C++ implementation of the behavior descriptions shown in Figure 1b and 1c. Each process instance is associated with only one *RTDS_Scheduler*.
- *RTDS_MessageHeader* encapsulates SDL-RT messages. It includes some additional information required for handling communication between processes. The *sender* and *receiver* attributes are unique identifiers of the source and destination process instance.
- *RTDS_TimerState* implements the SDL-RT timer. The core functionality is given by the *watchDogId* attribute, which is a ns-3 timer. This is an important concept because it provides integration with the ns-3 library. All events are handled by the the ns-3 scheduler and there is no need for synchronization or external mechanism to interact with the simulator.
- *RTDS_Trace* implements the tracing mechanism used for visualization (see Section 4).

SDL-RT deployment descriptions (Figure 2) are used to automatically generate a configuration for the ns-3 simulator. The generated code for the client-server example is shown in Listing 1.

First, code is generated for all *nodes*: ≪node≫ (Lines 3-8), ≪device≫ (Lines 10-18), and ≪channel≫ (Lines 20-21). Each ≪node≫ includes an implicit creation of its associated *RTDS_Scheduler* and the *RTDS_Env* environment process, which is used for distributed communication (see Section 2.2).

Code generation continues wit *components* (*RTDS_Proc* instances) and their *dependencies*. Dependencies associate the *RTDS_Proc* instance with the ≪node≫ through its implicit *RTDS_Scheduler* (Lines 23-24).

Finally, all *connections* are implemented: ≪node2device≫ (Lines 26-27) and ≪device2channel≫ (Lines 29-30). All code in SDL-RT comments (*Configuration* rectangle in Figure 2) is placed after the generated code.

```
1  int main(int argc, char **argv)
2  {
3    Ptr<Node> clientNode = CreateObject<Node>();
4    RTDS_Scheduler clientNode_scheduler(clientNode);
5    RTDS_Env(&clientNode_scheduler);
6    Ptr<Node> serverNode = CreateObject<Node>();
7    RTDS_Scheduler serverNode_scheduler(serverNode);
8    RTDS_Env(&serverNode_scheduler);
9
10   Ptr<PointToPointNetDevice> clientDevice = CreateObject<PointToPointNetDevice>();
11   clientDevice->SetAttribute("Address", Mac48AddressValue(Mac48Address::Allocate()));
12   clientDevice->SetAttribute("DataRate", DataRateValue(DataRate("5Mbps")));
13   clientDevice->SetAttribute("TxQueue", PointerValue(CreateObject<DropTailQueue>()));
14
15   Ptr<PointToPointNetDevice> serverDevice = CreateObject<PointToPointNetDevice>();
16   serverDevice->SetAttribute("Address", Mac48AddressValue(Mac48Address::Allocate()));
17   serverDevice->SetAttribute("DataRate", DataRateValue(DataRate("5Mbps")));
18   serverDevice->SetAttribute("TxQueue", PointerValue(CreateObject<DropTailQueue>()));
19
20   Ptr<PointToPointChannel> p2p = CreateObject<PointToPointChannel>();
21   p2p->SetAttribute("Delay", TimeValue(MilliSeconds(2)));
22
23   pClient(&clientNode_scheduler);
24   pServer(&serverNode_scheduler);
25
26   clientNode->AddDevice(clientDevice);
27   serverNode->AddDevice(serverDevice);
28
29   clientDevice->Attach(p2p);
30   serverDevice->Attach(p2p);
31
32   // Configuration
33   ...
34   return 0;
35 }
```

Listing 1: Generated code for the simulation configuration shown in Figure 2.

4 VISUALIZATION

Visualization is based on a set of traces produced by simulation execution. These traces are made available after the simulation is finished. This post-simulation visualization approach is also applied by popular network visualization tools like Nam and iNSpect. The simulation traces are a formatted representation of simulation events of interest related to distributed communication systems. The event types to be traced are organized as shown in in Figure 4.

The *RTDS_Trace* class implements the generation of the trace file. The list of traced events is accessible via *nodeEventList* and *networkEventList* attributes. There exists only one instance of *RTDS_Trace* in the simulation model and it is accessible by all *RTDS_Scheduler* and *RTDS_Proc* instances (see Figure 3).

The *RTDS_Node* contains information about the node (i.e., unique identifier, coordinates, etc.). This information is used to identify and display the nodes during visualization.

The *RTDS_NodeState* contains information about the node's state. The state of a node is actually an aggregation of all current states of process instances running on the node.

RTDS_NodeEvent and *RTDS_NetworkEvent* represent the traced simulation events. These events are visualized in two levels: node and network.

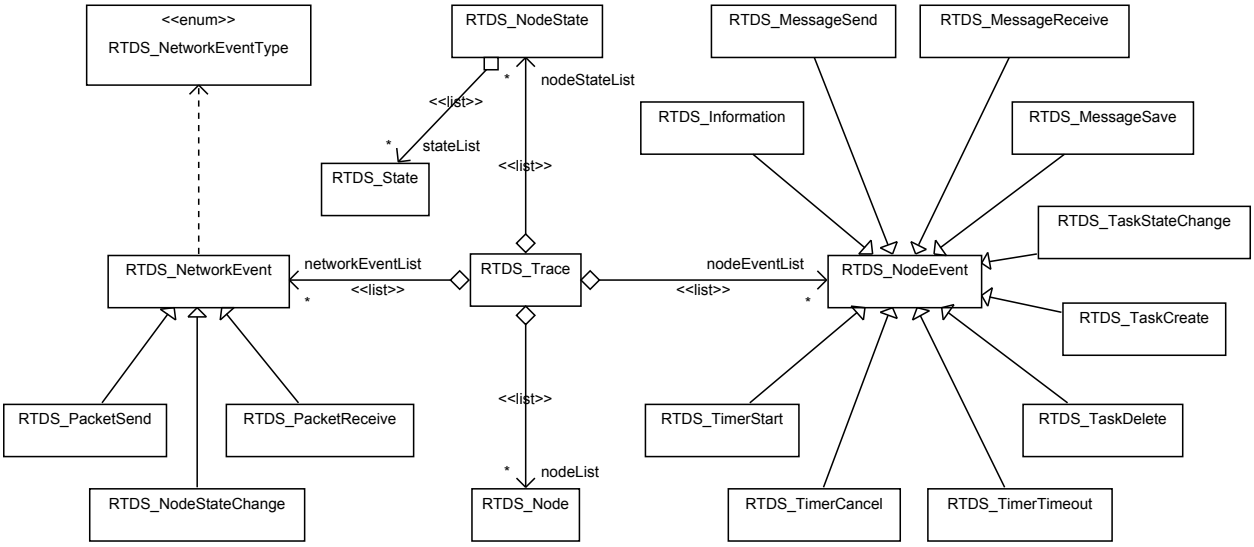


Figure 4: Class diagram for simulation trace generation.

4.1 Node Level

Message Sequence Charts are used for visualization at the node level. MSC is a language to describe the interaction between a number of independent message-passing instances (ITU-T 2011). It is a formal language with graphical notations. Its formal definition enables formal and automated validation. The textual form of MSC is mainly intended for exchange between tools and as a base for automatic formal analysis. MSC is often used in conjunction with other methods and languages. It can (and often is), for example be used in combination with SDL (ITU-T 1997).

MSCs describe the order in which events take place inside the node. These events are represented by the set of classes that extend *RTDS_NodeEvent* in Figure 4:

- *RTDS_TimerStart*, *RTDS_TimerCancel*, and *RTDS_TimerTimeout* represent SDL-RT timer events.
- *RTDS_MessageSend*, *RTDS_MessageReceive*, and *RTDS_MessageSave* represent SDL-RT message events. They are used to trace communication between process instances running on the same node.
- *RTDS_TaskCreate*, *RTDS_TaskDelete*, and *RTDS_TaskStateChange* represent SDL-RT process related events. The term task is intended as process instance.
- *RTDS_Information* can be used for debugging information.

The traces at node level are visualized using the MscTracer tool from PragmaDev (PragmaDev 2009).

4.2 Network Level

Communication between nodes and their state are displayed based on network visualization concepts. A node is represented by a filled circle with a unique identifier. The events that can be displayed are represented by the set of classes that extend *RTDS_NetworkEvent* in Figure 4:

- *RTDS_NodeStateChange* represents a change to the node's state. This type of event is displayed by changing the color of the node accordingly. A distinct color is assigned to each node state.
- *RTDS_PacketSend* represents a message sent from one node to another. The term packet is used only to make the distinction with *RTDS_MessageSend*, which is used to trace communication between process instances on the same node. This type of event is displayed by a named arrow from the sender to the receiver. Each message has a unique identifier.

- *RTDS_PacketReceive* represents a message received from a node. This type of event is displayed by removing the corresponding sent message (named arrow) from the view. This ensures correct visualization of communication events, because each message is displayed until it is received. If a sent message is never received, it is considered lost and it is displayed by a dotted named arrow from the sender to the receiver.

Even though these events are displayed in a network visualization fashion, the tracing format used to represent them is that of MSCs in textual form. This allows formal analysis and validation the system at both levels (node and network) with the standardized methodology defined in ITU-T 1997.

4.3 Tool for Visualization

We provide tool support for the visualization of events at the network level. Our tool (*DcsAnimator* - Distributed Communication System Animator) can interact with PragmaDev's MscTracer for visualization at the node level. Figure 5 shows a snapshot of the tool displaying simulation traces of the client-server example. Figures 5b and 5c show corresponding MSC traces for the nodes in Figure 5a.

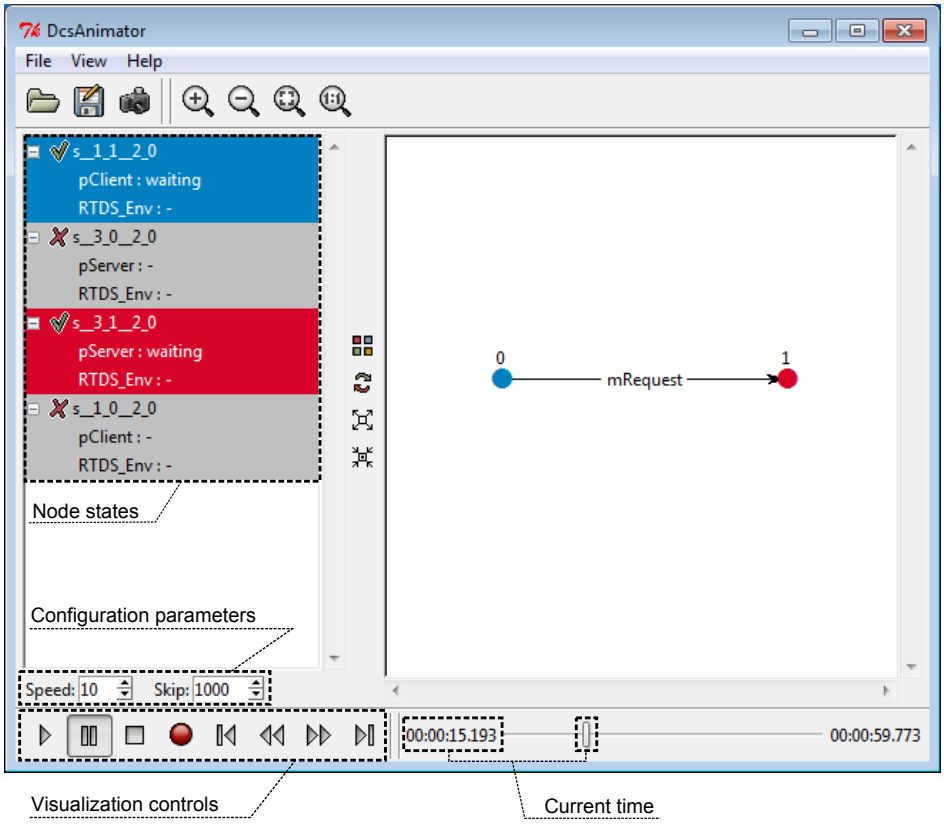
Time resolution is in milliseconds. *Current time* is synchronized between the tool and MscTracer. The visualization speed is expressed in percentage of real-time (100% means nearly-real-time visualization) and can be set in the *Configuration parameters*. *Visualization controls* can be used to navigate through traced events. Navigation can be time or event based. Time-based navigation means that the current time is increased (or decreased) by *Skip* milliseconds (Configuration parameters) and the events are visualized accordingly. Event-based navigation allows to jump to the next (or previous) event relative to the current one. The list of possible states is also configurable via the tool's interface. It is possible to choose which states to display during visualization. This becomes useful when the list is too long and there is no interest in displaying some of states.

5 CONCLUSIONS

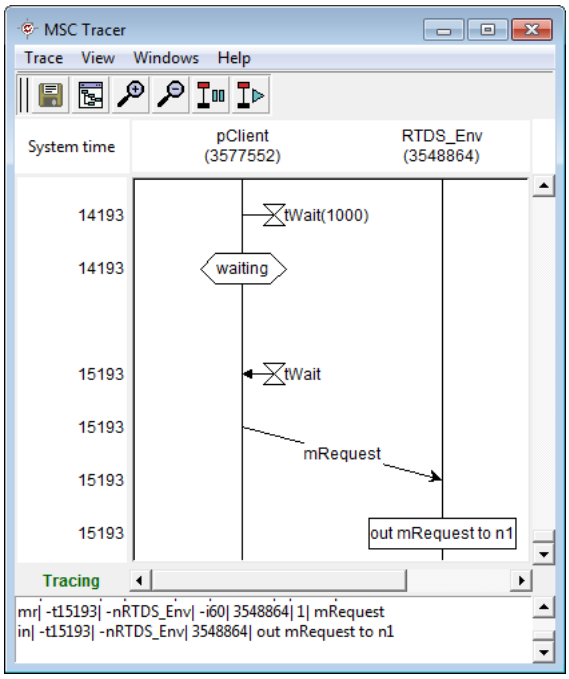
Network simulation and visualization is a well known and popular approach to analysis and validation of distributed communication systems. Nevertheless, the description of simulation models is not a trivial task. The models are hard to maintain and simulator dependent. Also, network visualization does not offer the required level of detail when nodes' internal behavior is concerned.

The approach presented in this paper uses SDL-RT and MSCs to address this issues. SDL-RT is used to describe simulation models and configurations. Implementation for the ns-3 simulator can be automatically derived from these descriptions with the help of code generation. The models for code generation are flexible enough to be easily adapted for other network simulators. The only limitation is the programing language; it has to be C/C++ because this is the action language of SDL-RT.

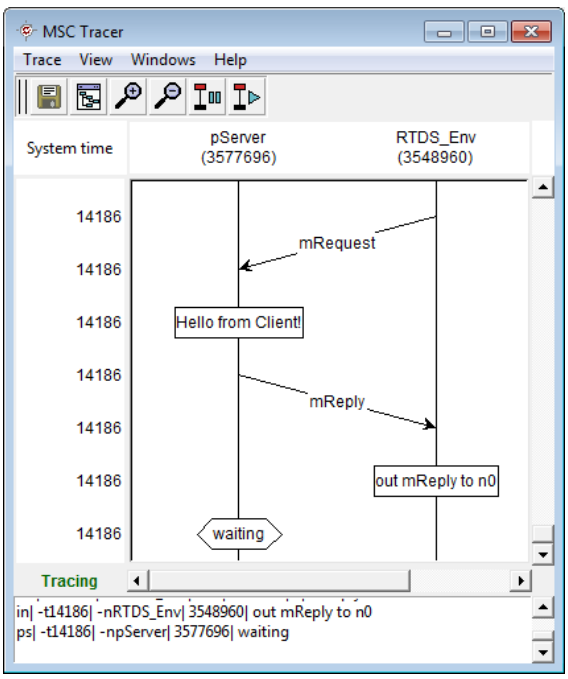
We use MSCs in combination with network visualization for displaying nodes' internal behavior. Our tool support provides detailed visualization of system's dynamics at network and node level. The format used for simulation traces is that of MSCs in textual form. This allows formal analysis and validation of the system at both levels.



(a) Visualization at the network level with DcsAnimator.



(b) Client node in MscTracer.



(c) Server node in MscTracer.

Figure 5: Snapshot of the visualization tool.

REFERENCES

Ahrens, K., I. Eveslage, J. Fischer, F. Kühnlenz, and D. Weber. 2009. "The Challenges of Using SDL for the Development of Wireless Sensor Networks". In *SDL 2009: Design for Motes and Mobiles*, edited by R. Reed, A. Bilgic, and R. Gotzhein, Volume 5719 of *Lecture Notes in Computer Science*, 200–221. Springer Berlin / Heidelberg.

Belue, J. M., S. H. Kurkowski, S. R. Graham, K. M. Hopkinson, R. W. Thomas, and J. W. Abernathy. 2008, December. "Research and Analysis of Simulation-based Networks through Multi-Objective Visualization". In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, 1216–1224. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Blunk, A., M. Brumbulli, I. Eveslage, and J. Fischer. 2011, July. "Modeling Real-time Applications for Wireless Sensor Networks using Standardized Techniques". In *SIMULTECH 2011 - Proceedings of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, edited by J. Kacprzyk, N. Pina, and J. Filipe, 161–167: SciTePress.

Breslau, L., D. Estrin, K. R. Fall, S. Floyd, J. S. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. 2000. "Advances in Network Simulation". *IEEE Computer* 33 (5): 59–67.

Brumbulli, M., and J. Fischer. 2011. "SDL Code Generation for Network Simulators". In *System Analysis and Modeling: About Models*, edited by F. Kraemer and P. Herrmann, Volume 6598 of *Lecture Notes in Computer Science*, 144–155. Springer Berlin / Heidelberg.

Doldi, L. 2003. *Validation of Communications Systems with SDL: The Art of SDL Simulation and Reachability Analysis*. Wiley.

Estrin, D., M. Handley, J. S. Heidemann, S. McCanne, Y. Xu, and H. Yu. 2000. "Network Visualization with Nam, the VINT Network Animator". *IEEE Computer* 33 (11): 63–68.

Fischer, J., J.-P. Redlich, J. Zschau, C. Milkereit, M. Picozzi, K. Fleming, M. Brumbulli, B. Lichtblau, and I. Eveslage. 2012. "A wireless mesh sensing network for early warning". *Journal of Network and Computer Applications* 35 (2): 538–547.

Gotzhein, R., and P. Schaible. 1999. "Pattern-based development of communication systems". *Annals of Telecommunications* 54:508–525.

Henderson, T. R., S. Roy, S. Floyd, and G. F. Riley. 2006, October. "ns-3 Project Goals". In *Proceeding from the 2006 workshop on ns-2: the IP network simulator (WNS2 '06)*, edited by T. Jiménez and D. Ros. New York, NY, USA: ACM.

ITU-T 1997. "SDL+ methodology: Use of MSC and SDL (with ASN.1). ITU-T Recommendation Z.100 - Supplement 1". Technical report, International Telecommunication Union.

ITU-T 2007. "Specification and Description Language (SDL). ITU-T Recommendation Z.100". Technical report, International Telecommunication Union.

ITU-T 2011. "Message Sequence Chart (MSC). ITU-T Recommendation Z.120". Technical report, International Telecommunication Union.

Kuhn, T., A. Geraldy, R. Gotzhein, and F. Rothländer. 2005. "ns+SDL The Network Simulator for SDL Systems". In *SDL 2005: Model Driven*, edited by A. Prinz, R. Reed, and J. Reed, Volume 3530 of *Lecture Notes in Computer Science*, 1166–1170. Springer Berlin / Heidelberg.

Kurkowski, S., T. Camp, N. Mushell, and M. Colagrosso. 2005, September. "A Visualization and Analysis Tool for NS-2 Wireless Simulations: iNSpect". In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, edited by G. Riley, R. Fujimoto, and H. Karatza, 503–506. Washington, DC, USA: IEEE Computer Society.

OMG 2011. "OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1". Technical report, Object Management Group.

PragmaDev 2009. *MSC Tracer User Manual*.

Riley, G. F. 2003, October. "The Georgia Tech Network Simulator". In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research (MoMeTools '03)*, edited by G. Carle, H. Ritter, and K. Wehrle, 5–12. New York, NY, USA: ACM.

Schaible, P., and R. Gotzhein. 2003. "Development of Distributed Systems with SDL by Means of Formalized APIs". In *SDL 2003: System Design*, edited by R. Reed and J. Reed, Volume 2708 of *Lecture Notes in Computer Science*, 158–158. Springer Berlin / Heidelberg.

SDL-RT Consortium 2006. "Specification and Description Language - Real Time. Version 2.2". Technical report, SDL-RT Consortium.

AUTHOR BIOGRAPHIES

MIHAL BRUMBULLI is a PhD student in computer science at Humboldt University Berlin, Germany. He received a diploma in computer engineering from the Polytechnic University of Tirana, Albania. His research interests are in formal description techniques for modeling and simulation of distributed systems. His email address is brumbull@informatik.hu-berlin.de.

JOACHIM FISCHER is a professor for System Analysis, Modeling and Computer Simulation at Humboldt University Berlin, Germany. His major research interests are object-oriented modeling of dynamic systems, especially of distributed systems and the development of simulation tools. The combination of model checking, simulated execution and target code generation based on abstract system models plays an important role. He is a member of OMG and SDL-Forum. Currently he is the speaker of an interdisciplinary graduate school supported by the German Research Foundation (DFG). His email address is fischer@informatik.hu-berlin.de.