Managing cross layer information in OMNeT++ network simulations

Laura Marie Feeney
Swedish Institute of Computer Science and Uppsala University
lmfeeney@sics.se

***Abstract*—This paper describes a general approach to dealing with cross layer information in OMNeT++-based network simulations. Rather than prescribe a specific cross layer architecture, this work is intended to address the software engineering problem of passing information between simulation modules in a sound way. An XML-based mechanism for passing information between modules using OMNeT++'s `controlInfo` data structure is presented. Because XML is an intermediate format for the OMNeT++ network description language (NED), it is further suggested that allowing NED files to be annotated with information about cross layer interactions will enhance readability and reuse-ability. A further goal of this paper is to initiate discussion and collaboration in the Swedish (and international) OMNeT++ community.**

I. Introduction

Cross layer optimizations have been widely studied for communication networks, especially wireless networks. Examples of such optimizations include channel-aware ad hoc routing protocols, content-aware encoding of video streams, and application-aware power management protocols. There has also been considerable effort directed toward the design of architectures for managing cross layer information. Many of the key challenges in developing general purpose cross layer architectures relate to the practical context, where multiple optimizations may interact in unexpected ways and where maintainability and interoperability are of paramount concern.

There has been a corresponding need for simulation environments that support evaluation of cross layer techniques. It is perhaps not too surprising that there are corresponding difficulties in simulating cross layer optimizations in such that the resulting code is maintainable and can be (re) used to model interactions between independently developed optimization strategies.

This work therefore presents an extension to the OMNeT++ [10] simulation environment that is intended to address the *software engineering* problem of passing information between simulation modules in a robust and flexible way. It is particularly suitable for situations where the modular structure of the simulation software does not exactly mirror the layered structure of protocol stack, where it is desirable to use a combination of independently developed simulation modules, or where potential interactions between cross layer optimizations are of interest.

The paper describes limitations of the existing OMNeT++ mechanisms for passing information between simulation modules and presents a prototype implementation for treating the `controlInfo` data structure as a structured XML element that contains the inter-module information associated with a given message. The choice of XML is motivated by OMNeT++'s existing support for XML as an intermediate format, as well as broad knowledge and acceptance of this format.

The paper also suggests that OMNeT++ network description (NED) language (which already supports XML as an intermediate format) be extended to include XML descriptions of all information that is passed between modules. Such annotation would make it easier for the user to identify potential conflicts between cross layer mechanisms, especially when combining or refactoring independently developed modules. It would also make it possible to automatically generate code for some minimal validation of simulation operation, as well as for providing various utility routines.

II. Background

Omnet++[10], [8] is a modular discrete event simulation environment which has been used as the basis for a number of network simulation tools, including the mobility-framework[2] and the INET [6] framework.

In these simulation frameworks, OMNeT++'s NED network description language is used to define each network host as a collection of modules, which are connected to form a directed multi-graph. Simulation messages (representing both data and internal control) are passed along edges of the graph.

A communication data frame is generally represented by an OMNeT++ message that passes through a sequence of modules in the host, much as it does in the layered protocol stack. Connectivity between hosts can be modeled using direct links. For wireless networks, however, a specialized channel module is generally used to model propagation effects on the shared wireless channel.

In addition to the message data itself, there are two kinds of information that can be passed between modules: The first is information associated with a message that is passed from one module to another via `send(msg, ..., controlInfo)` interface. The `controlInfo` parameter is a generic pointer (`cPolymorphic`), which must be appropriately interpreted by the receiver. The OMNeT++ documentation gives as an example of this usage passing TCP/IP header information between the transport and network layer modules. It is limitations of this interface that this work mostly intended to address.

The second kind of information is associated with a module, rather than with a message that is passed between modules.

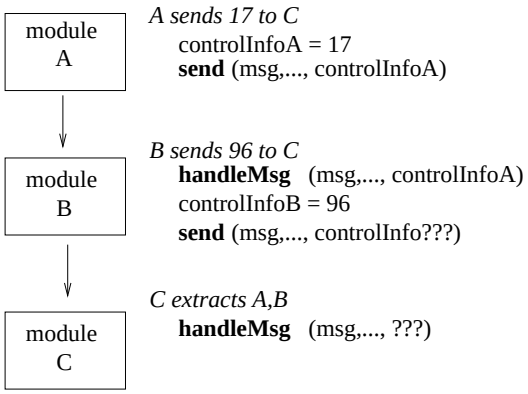

Fig. 1. There is no common mechanism to pass information across modules that are not directly connected.

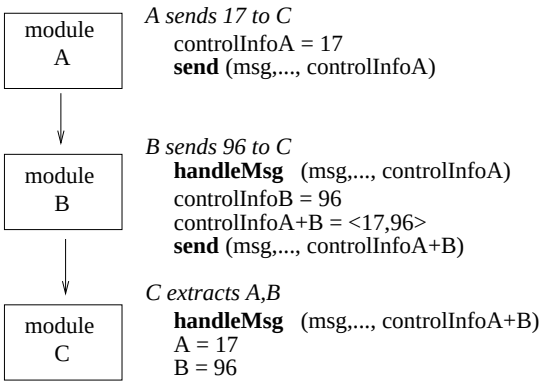

Fig. 2. Ad hoc mechanisms are unsatisfactory, as they result in code that is difficult to maintain.

Such information can range from simply reading a state variable (such as residual battery capacity) to initiating a complex, asynchronous interaction (such as an on-demand route discovery). Some existing mechanisms for dealing with this case are discussed briefly in section IV, but the problem is too general to be amenable to a single solution.

III. Message metadata

In keeping with the idea of an cross layer architecture "agnostic" model, the proposed mechanism is intended to solve the problem of passing metadata information associated with a message between simulation modules in a relatively sound and user-friendly way. This separates the problem of software engineering in the simulation environment from the problem of designing a cross layer architecture for a specific context.

In this context, the `controlInfo` model's main flaw is that it only allows one piece of information to be passed between simulation modules that are directly connected via a gate. There is no common mechanism for allowing information to be passed between modules that are not directly connected or for multiple modules to pass information to the same module.

Consider the (simplified) example in Figure 1). Modules A and B both associate metadata intended for use by module C with a message that is sent along the marked edges of this module graph. When module B `handle`'s the message, the `controlInfo` parameter already refers to metadata set by module A, so module B cannot simply set the `controlInfo` parameter to refer its own metadata.

In order to pass both pieces of metadata to module C, module B has to create a `controlInfo` structure that contains the metadata information from both A and B as in Figure 2. Further complexity arises if new metadata is added to a module, or if there are multiple paths though the code for modules A and B, such that different paths result in the presence of different metadata.

In general, such ad hoc solutions are unsatisfactory, because they require that each module take into account metadata that previous modules may have included in the `controlInfo` when creating an aggregated structure in which to include its own metadata or when extracting metadata. These inter-dependencies make it difficult to re-factor or re-use modules, significantly reducing the usefulness and maintainability of the resulting code.

A. Extending controlInfo

To address the limitations of the `controlInfo` mechanism, there needs to be some way to allow the `controlInfo` to accumulate structured information as the message is passed from module to module. In addition, this model has to be enforced (at least by convention) in all modules through which annotated messages pass. One obvious way to do this would be to re-implement `controlInfo` as an indexed list of `controlInfo` elements, which each module could add to or search and retrieve from as needed. Such a structure would be not unlike an XML document. In fact, XML is a natural mechanism for passing a structured and extensible collection of data. Moreover, XML is a widely known data description format and OMNeT++ already supports XML for various file formats (NED, msg and parameter files) and a simple API for manipulating XML fragments is provided via `cXMLElement` class.

Therefore, rather than create an ad hoc extension to the `controlInfo` data structure, we re-define `controlInfo` to contain a `cXMLElement` that is the root of an XML fragment that accumulates any metadata associated with the message as it passes between modules. Each piece of information that is intended for cross module access is added to the XML document with an appropriate tag. The tag is made up of two parts, the module path (its `FullPath()` in the simulation host) and a module local name for the information. To read the metadata, a module need only know the path to the module in the host structure (which can be configured as a module parameter in the Host NED file) and its name and datatype (float, int, bool). If the simulation runs on a single machine, the value may be stored as an untyped data buffer for better performance, rather than be serialized. (In the case of parallel simulation, all messages and metadata must, of course, be serialized.)

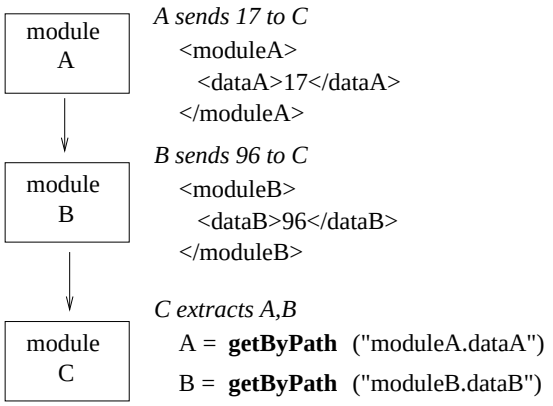

Fig. 3. The previous example, using XML fragment to contain metadata. Modules A and B add tagged data. Module C extracts data by tag.

Once the message is placed on the channel, the `controlInfo` is deleted, as only information actually present in a network frame's payload can obtained by the remote host.

This approach makes module code more maintainable, because each pair of modules can add and extract metadata information independently of other metadata and modules. This also allows for greater flexibility in implementing modules, since there need not be a directly mapping between software modules and layers of the network architecture. Modules can be re-factored into smaller submodules with requiring custom `controlInfo` data structures in each intervening module. (It is actually this case that motivated this work). Moreover, the use of XML data tags makes code more readable and provides additional information to OMNeT++'s visualization interfaces.

This mechanism can be used locally within any (locally cooperating) set of modules; a prototype set of interfaces has been implemented.

B. Extending NED

Unexpected interaction between cross layer optimizations is a significant problem for their use in many practical contexts, particularly environments where multiple protocols and applications are present. How such conflicts should be resolved is a problem of cross layer architecture that is far outside the scope of this work.

The proposed mechanism is intended to enable users of OMNeT++-based simulation frameworks to implement modules in a way that they can be maintained and re-used: a capacity that is particularly advantageous in an open source environment. This flexibility also makes it easier to build simulations that incorporate several, possibly independently developed, optimization techniques, as well as to avoid doing so inadvertently.

Consider (as an extreme example) a wireless host that is composed of independently developed modules, some of which make use of cross layer information. The protocol stack includes: a routing protocol that sets the transmit power based on neighbor density; a power save protocol that sets the transmit power based on the remaining battery capacity; and a MAC layer that sets the transmit power based on its estimate of channel conditions at the receiver. The simulation results are likely to be unexpected, at best. Highlighting the use of cross module information in the NED files – the most user-visible description of a simulation module – may be helpful in this regard.

The internal description files used in OMNeT++ already support XML as an intermediate format. It is therefore proposed that the NED language be extended to include an XML description of the out-of-band information that is passed between modules in the `controlInfo` structure. In particular, each `out:` gate should include the XML tag(s) of any data that is added to the `controlInfo` data structure.

This does not prevent the kinds of conflicts described above, but it does make potential interactions more obvious to the user. The NED files are the most compact and user-friendly description of the modules and their interactions; they are far easier to read than C++ internals. In our example, even if the transmit power information is labeled with different XML tags in each module, a user will probably see that the combination of modules with `out:` gates with tags `TxOut`, `PowOut`, and `TxPower` (for example) is a likely source of trouble.

Furthermore, because the NED files contain the description of the module structure, it may be possible to generate various kinds of utility code. One possibility is to generate debugging code to validate whether `controlInfo` sent from the relevant gates is actually present. Alternatively, it may be possible to generate code to insert and extract XML elements from the `controlInfo` structure. This possibility opens questions as to whether the XML descriptions should include type information (as in the OMNeT++ .msg format) or whether module-local code should interpret polymorphic data.

This extension requires substantial addition to the public NED interfaces, not to mention non-trivial programming effort. No prototype has been developed, as further discussion is required within the community. (Note that the `controlInfo` mechanism described in the previous section does not depend on NED extension.)

C. Performance

This section discusses two aspects of the performance of the proposed scheme: the first is re-usability and robustness and the second is run-time speed of the simulation. To the extent that computer time is cheaper than developer time, we focus on the former.

Omnet++ does not provide any mechanism for passing control info between other than between the module that `send()`'s and the module that `handleMessage()`'s the message. If an intervening module is added, or if one of the modules is re-factored and implemented as two separate module or if the module is re-used in a different context, then there is no longer any well-defined way to pass information between them.

In the proposed mechanism, only the module that creates a piece of metadata and any modules that access the data

need to have any knowledge of its existence and format. Access to cross layer information is *independent* of how the message traverses the module graph. As a result, the proposed mechanism is robust to nearly all changes in the host's module structure: modules can be re-used in new contexts, intervening modules can be added, removed or replaced, a module can be re-factored into submodules. The only change that is needed is the text string that modules use to access the data. (e.g. a module that uses `SomeData` from `subModuleZ` will only need to change `moduleA.subModuleZ/someData` to `moduleA.newSubModuleB.subModuleZ/someData`, if `moduleA` is re-structured). These path strings can even be specified as parameters in the relevant NED files, avoiding code changes altogether.

It is hard to overstate the software engineering complexity of building simulations with modules that need to pass metadata information between modules that are not directly connected, or in which there are multiple logical paths along which messages and their `controlInfo` is passed.

If a single `controlInfo` structure (data structure or class) is used, it must be able to reference all combinations of metadata that can be added to a message and provide methods for getting and setting it. This means that the code defining the `controlInfo` needs to be revised when modules are used in a new combination or context.

Keeping metadata local to each module and accessing it via a callback mechanism faces even greater complications: If an intervening module implements buffering, then there may be more than one message present in the system. This means that the value of the crossLayerInformation for each message has to be buffered in the source module until it has been accessed by all reading modules. To operate correctly, the source module must therefore take into account the internal implementation of any intervening modules (e.g. timeouts, potential reordering of messages) to ensure that it correctly caches the right metadata values for each message.

However, using XML based controlInfo does pose a significant performance penalty: it is about a factor of eight times slower to use XML-structured `controlInfo` to set and get metadata, than it is to simply pass the data directly via an untyped `controlInfo` buffer.

Note that this increase reflects only the cost of using `controlInfo`: In this case, the simulation modules are only setting and getting control information. In practical simulations, the effect will be considerably less, because other simulation processing, particularly mobility modeling and simulation of wireless propagation and interference, is a significant cost. Moreover, the `cXMLElement` implementation in OMNeT++ is currently not optimized. An optimized implementation, particularly one specialized for using with `controlInfo`, might provide much better performance.

IV. Module state information

A second kind of cross layer information is not associated with a particular simulation message, but is maintained by a simulation module. Examples of such information include the state of some hardware component of the host (such as a radio receiver or battery) or some data structure that is maintained and updated by the host (such as a neighbor table). There are a broad range of existing mechanisms for handling these cases; the most complex do not seem amenable to the simple strategies outlined above.

Existing frameworks explicitly support synchronous notification mechanisms such as the mobility-fw's `Blackboard`, which provides a interface such that a callback routine in one or more subscriber modules are invoked when when an event is published.

By contrast, there is no single mechanism for one module to read the state information of another module. An appropriate sequence of `parentModule()` and `submodule(name)` calls (or even `cSimulation::moduleByPath()`) can be used to access any simulation module. Given a module, a caller can access any public data and methods of the associated C++ class. Such an interaction could be quite simple (e.g. `Battery::getResidualEnergy()`) or quite complex, as in the case of activating significant functionality in the called module.

Alternatively, if there is an explicit connection between the two modules, a pair of request-reply messages can be used to exchange information. Even if reply involves no delay, the reply message may not be guaranteed to be the next message that is received (on some gate) by the requester and some kind of buffering is needed. Again, in more complex cases potentially involving long delays (e.g. on-demand route discovery), the logic for buffering and timeout may be quite complex.

V. Related work

In addition to being a potentially useful bit of software engineering, the proposed mechanism differs from some existing solutions in focusing on allowing information to be shared between simulation modules in a way that promotes maintenance and reuse. It is cross layer architecture "agnostic" and hence very general, while at the same time providing a robust structure and good readability. Related work found in various open source simulation environments is discussed below.

The PAWiS [4] project has developed an OMNeT++ framework for cross-layer optimization in sensor networks. The work seems to address the problem of modeling interaction between multiple optimization parameters by performing refinement cycles to find configurations and parameters that give good performance. Little has been published about the internals of this system.

Ns-2 [3], [7] remains the most widely used network simulation environment, particularly in academic research environments. There are at least two efforts to provide support for managing cross-layer information in ns-2.

The Multi-Interface Cross Layer Extension for ns-2 (MIRACLE) [1] is intended for simulation of next generation communication networks, especially multi-radio wireless systems. MIRACLE defines a cross layer communication architecture

where multiple instantiations of the same functionality (e.g. multiple radio interfaces) may present in a given layer. All modules are connected to a message bus, the `NodeCore`, and cross layer processing is performed by `PlugIn`'s attached to the bus.

Informal reports[9] describe extensions to ns-2 that simulate a general purpose cross-layer architecture, which is described in terms of a matrix of "knobs" passing control down and "dials" passing information up. Little has been published about the internals of this system.

Ns-3 [5], [7] is the successor to the ns-2 simulation environment and is still very much still under development. In the current version of ns-3, each packet is modeled as a byte buffer representing the packet's serialized (wire) format and a set of fixed-size tags that can be associated with some or all of the bytes in the packet. To date, there does not seem to be any structured convention for using tags.

Allowing a tag to refer to a subset of byes is intended to help address the problem of how tags are handled across fragmentation and re-assembly (though some issues still seem to be open). In fact, the proposed OMNeT++ mechanism has similar issues: For example, consider a network packet that is fragmented at the MAC layer and a different received signal strengths (RSS) are observed for each frame at the receiver. It is not clear how the RSS value(s) would be represented in the de-fragmented packet that is passed on to a higher layer protocol (e.g. a routing protocol that used signal strength information in its routing decisions).

In general, the mechanism proposed in this paper seems to be similar in spirit to the tags mechanism proposed in ns-3, but takes advantage of the structured environment provided by XML. It should be emphasized that, as some of these systems are actively under development, relatively few formal references are available. This discussion therefore includes a substantial amount information from informal sources, like web documents, mailing list archives, and code documentation, which may quickly become out of date.

VI. Future work

The paper presented a simple XML-based mechanism for using OMNeT++'s existing `controlInfo` and `cXMLElement` structures to provide a flexible way to allow information to be passed along with messages traversing the OMNeT++ modules defining a protocol stack. This convention can be adopted fairly easily in a network simulation package, without modification to underlying the OMNeT++ software.

The more general extensions that have been proposed require changes to support annotating modules and gates in the NED file format, which is OMNeT++'s most visible public interface. Further discussion in the OMNeT++ community is needed to determine whether there is sufficient acceptance of the general principle of using XML to managing cross module information passing to warrant further activity.

Another possible extension of this approach is data collection, which another natural target for passing and storing information in XML format. Currently, the `controlInfo` is deleted from a packet when it reaches the channel and passes to another host, since information can be passed from one host to another only if it is included the transmitted packet. However, it might be interesting to consider the possibility of providing a similar, persistent XML-based structure for data collection, especially for data that is naturally multihop, such as end-to-end delay.

VII. Conclusion

This paper describes a mechanism to better support passing metadata information between modules in OMNeT++ based network simulations. The `controlInfo` parameter is defined as an XML fragments, which allows metadata to be passed between arbitrary modules in a structured manner. The modification to the `controlInfo` data structure can be used locally among a group of cooperating modules. In addition, it is suggested that NED files be annotated with a corresponding XML description, making relevant interactions more visible to the user.

Providing better support for passing information between OMNeT++ modules make it easier to develop and re-use modules in a sound and maintainable way. Such support potentially makes it easier to study cross layer protocols and architectures, even in the case of complex interactions.

It is hoped that this paper will initiate discussion and collaboration in the Swedish (and international) OMNeT++ community.

VIII. Acknowledgments

Parts of this study were carried out within the VINN Excellence Center WISENET, partially funded by VINNOVA, the Swedish Governmental Agency for Innovation Systems.

References

[1] N. Baldo, F. Maguolo, M. Miozzo, M. Rossi, and M. Zorzi. ns2-MIRACLE: a modular framework for multi-technology and cross-layer support in network simulator 2. In *ValueTools '07: Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, 2007.

[2] W. Drytkiewicz, S. Sroka, V. Handziski, A. Koepke, and H. Karl. A mobility framework for OMNeT++. In *3rd International OMNeT++ Workshop*, Jan. 2003.

[3] K. Fall and K. Varadhan, editors. *The ns Manual*. The VINT Project, 2000-2008.

[4] J. Glaser and D. Weber. Poster abstract: Simulation framework for power aware wireless sensors. In *Fourth European Wireless Sensor Networks Conference, (EWSN)*, Jan. 2007.

[5] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. B. Kopena. Network simulations with the ns-3 simulator. Demo at SIGCOMM, 2008.

[6] INET framework for OMNeT++. http://www.omnetpp.org/doc/INET.

[7] The nsnam homepage. http://www.nsnam.org.

[8] The omnet++ homepage. http://www.omnetpp.org.

[9] S. Varatharajan, A. Jabbar, A. Mohammad, R. Naidu, J. P. Rohrer, P. Upadhyay, and J. P. Sterbenz. Cross-layer framework for the ns-2 simulator. ITTC IAB poster (unpublished), University of Kansas., June 2008.

[10] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, June 2001.