Simulation and Implementation of an E-Commerce Communications Infrastructure using XML Specifications

Simon R. Chudley
Dept. of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ
United Kingdom
`src299@ecs.soton.ac.uk`

Ulrich Ultes-Nitsche
Dept. of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ
United Kingdom
`uun@ecs.soton.ac.uk`

Abstract

We report in this paper on a project dealing with the development of an XML-based tool for the management of the communication infrastructure within an e-commerce system. The major aim is to be able to simulate any given infrastructure prior to its implementation in order to detect potential problems early in a cost-effective way. We are focusing mainly on security problems that may arise. From the so tested XML description of the network, the settings of the nodes on the given physical network are generated automatically from the XML. As there are no assumptions made about the underlying network, our approach works for any kind of e-commerce environment, including mobile e-commerce systems.

1. Introduction

The major success factor for e-commerce systems obviously is customer satisfaction [Turban 2000]. Besides the visual (web-)presentation of a company and ease of use of web-interfaces [Ince 2002], [Deitel 2000], the network infrastructure plays a significant role in establishing customer satisfaction. The network must perform well even during peak hours of usage and the network security must be constantly fully functioning to ensure the customers' trust is maintained.

We report in this paper on an ongoing project developing a tool for designing an e-commerce network infrastructure using XML [McLaughlin 2000], [XML]. The tool enables us to simulate the network for different settings of parameters prior to its implementation. This increases our confidence in the correct functioning of the designed network, including security settings such as firewall rules. The XML design is such that it can be implemented automatically by compiling the XML directly into values of variables of the network nodes. Since we do not make any assumptions on the type of network, our approach will be applicable to highly heterogeneous environments (also called multi-channel systems; see [Carignani 2000]) that are typical for e-commerce application: In a business-to-business setting (B2B), for instance, none of the involved companies will be willing to apply significant changes to their physical network. So allowing heterogeneity in the network design is highly important, in particular when moving to various types of wired and wireless networks that will be core components in mobile e-commerce systems.

1.1 Driving Force

Establishing and managing network infrastructures between distributed nodes can be time consuming and risk prone. Such a statement is often re-enforced by the incorporation of new technologies, the merging of existing systems and the rotation of sub-systems. However, the functional requirements of e-commerce sub-systems remain constant, even though implementation specifics may differ greatly. The growth in distributed computing environments and the increase in complexity and interaction between nodes have lead system management away from the traditional *core-based* solutions. Multitudes of computational platforms are typically involved in e-commerce systems and even within one enterprise. Managing the behaviour at node level of such systems, and allowing ease of modification can require additional expertise, even though the overall task remains the same.

The driving force behind this paper is to investigate the addition of an extra level of abstraction between the operational requirements and implementation of such nodes within e-commerce environments. Such an abstraction has the advantage of being sufficiently withdrawn from specific node level details to allow sub-systems to be interchanged, whilst maintaining the required behaviour. Utilising these high level descriptions, being specified in a generic syntax (XML), a scope for simulation is introduced. Automated analysis can attempt to model the interactions between nodes and sub-systems, leading onto validation of security, performance and scalability.

2.   Conceptual Overview

One such situation that this approach applies to is the management of network nodes on a local area network. Typically, within these environments different implementations of sub-systems are used, mainly for reasons of security, performance and fit for purpose. We are referring here to a single e-commerce sub-system. Such implementations may differ by being based on varying operating systems, or using an 'off the shelf' solution. Due to this, the processes undertaken to manage such solutions differ, leading to complications in the implementation and maintenance of these services.

For example, a node running Linux to provide a Domain Name System (DNS) [DNS] service may use differing configuration files from a functionally similar FreeBSD solution [FreeBSD]. However, for overall system management, this level of detail is not required, the main emphasis is on the behaviour of the standardised DNS protocol, which is therefore the same over the two implementations.

Translations are to be constructed describing the process of mapping between the standardised abstracted description and, in the previous example, the configuration files for the two DNS implementations. Utilising these descriptions and translations, the two nodes could be interchanged, whilst still maintaining the desired behaviour of that sub-system.

Maintaining the local area network paradigm, there are varieties of situations where this concept could be applied. Due to the requirements of networks, some of which being performance and security, they often consist of nodes running different operating systems and different implementations of standardised protocol services. Often within one operating system there may be various different solutions to a common goal, offering different levels of performance for example. However, due to the knowledge and time required to convert configuration files between these solutions, businesses tend not to *experiment* with the inferior solution. With the proposed layer of abstraction in place, and translations from this to the two solutions, it will enable the business to switch implementations with little effort. In addition, simulating the XML model prior to the system's implementation can test solutions to the same e-commerce sub-system.

The abstracted description of sub-systems may not stop at individual nodes. Higher level interactions could be modelled, the network in this case. Modelling the links between sub-systems and nodes would allow for an overall picture of the enterprise. Using this information and the details on the nodes, an analysis or simulation could be carried out to assess various characteristics of an entire e-commerce network infrastructure. A graphical representation of the system's conceptual overview is given in Figure 1.
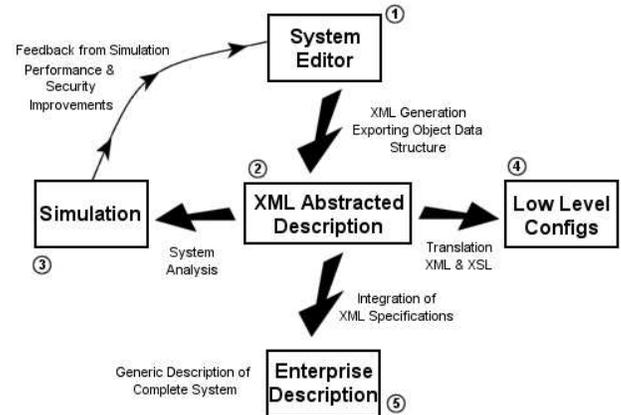

Figure 1. Conceptual view of the proposed system

One prime candidate for simulation is the situation involving filter based firewalls. These have the responsibility of protecting sub-systems, typically used at major network edges. Being able to model the behaviour of a firewall, using the abstracted descriptions mentioned previously, would be a great asset. Packets of data could be simulated as they progressed through the network, allowing the security of the system to be analysed without being concerned with the implementation level constructs.

3.   Describing the Abstraction

The primary objective to achieving this common goal is to be able to describe the services using a common language. The technology of choice for this is XML [McLaughlin 2000], [XML]. Extensible mark-up language provides a mechanism of describing services in an abstracted manner, using the validation techniques that come with XML to enforce these conventions. The XML model is the core component of our tool (number 2 in Figure 1).

Elements within the system, such as network services and nodes are described using a section of generic entities and values, conforming to the XML Schema for that specific entity. Such descriptions should be able to map the functionality from each implementation configuration.

XML configuration may either be a sequence of rules, such as in the firewall service example, or a list of static statements. Both approaches must be of sufficient abstraction to ensure that they are capable of describing all lower level implementations. Hence, using the firewall example, the generic firewall XML description must be able to be converted into configuration for a FreeBSD machine and a Linux machine.

To design a new system from scratch or alter an existing layout, the system editor (number 1 in Figure 1) provides all functionality needed to generate the XML models. The simulation tool (number 3 in Figure 1)

provides means of offline system testing and feedback of improvements.

4.   Service Constructs

A logical grouping of service configuration can be described using the concept of a service construct. The concept of rules being *logically equivalent* is not defined by this project, but used purely as an element of the system itself. An example of such a grouping is a set of rules within a firewall to enable a virtual private network to be established. Within this, there may be six rules for example to describe the required functionality, applying to two specific nodes. Variable mappings, introduced in the next section, can be used to control the behaviour of such service constructs.

In addition to this, service constructs can be defined within an external library. This allows for collections of commonly used, logical groupings of service configuration to be made available to all nodes within the system. Such constructs can be referenced from within the service XML via the central repository, providing yet a higher level of abstraction from the standard rule descriptions.

5.   Variable Mappings

On any one node within the system, configuration details are often closely related between the actual services. For example, on a network node, the IP addresses of the network interfaces may be required in both the Firewall and the Network Address Translation services. There is also the requirement for lexical variable scoping, applying throughout the hierarchy.

Basic arithmetic within variables can be performed at service definition level, allowing flexible configurations to be produced. A simple example is with the definition of rule-based services. Typically, each rule will be associated with a given ID. However, this may not be known in advance, as the order of the constructs within the service may change, and references to external constructs within libraries may be used. Each logical block of configuration can assign ID's as offsets from some service level defined variable, allowing for more maintainable and flexible definitions.

Variable mappings are also important in the importing of external service constructs. External service constructs will define specific variables to control the section of rule's behaviour, such as IP addresses or nodes in the network example. These can be applied at any level within the XML description of the node itself, allowing lexical variable scope to be enforced.

6.   Configuration Level Function Calls

One of the primary aims of this project is to establish an architecture enabling managed, scalable configuration of network services. For this to be achieved various high level constructs were introduced, such as function calls.

Typically, the configuration of part of a node may depend on both local and remote services. A prime example is the situation where a service needs to operate on a node also running a firewall. For this service to operate as required, rules may need to be added to the firewall configuration, to allow specific ports accesses for example. A function call could be made from the firewall description itself, taking the intended target service as input, and dynamically generating the required firewall rules for that service to operate as expected.

A function call is defined as taking the XML output of a specific service construct (logical grouping of service configuration), feeding it through an XSLT processor (the function itself is an XSL style sheet [XSL]), and producing another XML service construct as output. A set of parameters can also be used to control how the function behaves. The final XML segment is directly substituted with the original function call itself, completing the evaluation process.

However, the service construct used as input to the function is not restricted to the local node or service. Indeed, a construct from any service running on any node within the system can be used, allowing for ease of information sharing. In addition, a whole service (hence all constructs within it) can be specified as input.

The ability to reference a whole service as input to a function greatly expands the abilities of the system. It is envisioned that the whole configuration of a single service could be based on the definitions of many others. The main advantage of this being that functions are evaluated at configuration generation time. All aspects of the input constructs are generated prior to evaluating the function itself.

A practical example of function calls is in the definition of a DNS (Domain Name System) service. Typically, sets of mappings from name to IP address, and IP address to name are maintained. However, there is an obvious dependency between both the forward and reverse lookups. Hence, adding a new node would require two updates to the configuration of the system.

Function calls can be used to solve this problem. A function definition is included within the DNS service configuration, and when evaluated, takes the forward (name to IP address) mappings as input. Using that XML definition it dynamically creates the reverse mappings, and returns the new XML DNS service construct. This is directly substituted with the original function call definition, allowing for standard processing to continue.

7.   Translation & Specialisation

After possibly several iterations of simulation and re-design of the abstracted descriptions, we have a system that we desire to implement. In our context, implementation is synonymous with translation (number 4 in Figure 1). Translation is the process of converting the abstracted XML descriptions for a service down to implementation level. This can be expressed as converting a firewall description, for example, into a sequence of firewall rules. These rules can then be applied directly to the firewall implementation in use, hence completing the transition from the high level description to platform level.

To achieve the translation, the technology of choice is XSL, extensible style sheets (*XSL Transformations*). These supply the basic mapping between the two layers of configuration and are specific to each implementation. For each end level configuration format, there will be two files. The first, an XML description, will provide a wrapper around the style sheet itself. This specifies lower level details about the translation, such as possible limitation of the conversion and overall style information. Details on the actual translations are stored within the XSL style sheet, giving the mappings between all abstracted constructs and their associated implementation level representations.

Prior to processing the service descriptions using the desired XSL style sheet, the system generates full and complete XML configurations for the require services. During this process we aim to generate a fully specified XML description of the service, hence resolving the dependencies between services introduced in sections 4, 5 and 6. Replacing all variable references by their associated values, pulling in external service construct definitions and evaluating functions achieve this. The result is a full definition of that service complying with the XML Schema, able to be directly processed by the XSLT translator.

In some instances, the translation may be on a rule-based approach, such as in firewall configuration. However, the translations must also support configurations that are purely lists of options and settings.

8.   Conclusion

This paper has reported on how the utilisation of abstracted XML descriptions within a network management tool could see great advantages. The XML descriptions would provide the infrastructure to be able to carry out automated analysis of sub-systems, and aid the integration of components to form a centrally managed enterprise solution (number 5 in Figure 1). The proposed architecture can be easily extended to cover new services,

and translations between implementations can be created and managed external to the system.

With an extra layer of abstraction, such as that provided by the XML, an overview of the whole system can be obtained. Such a rich resource of data could help in higher-level business operations, such as merging system functionality with others during business-to-business transactions. It also allows evaluating the security of the system on the XML-model-level against identified risks [Schoder 2000], [Redmill 2001].

Extending the ideas presented in this paper could lead to more synchronous e-commerce solutions. Various stages of the process described in Figure 1 could be fully automated, such as the translation and application of dynamically created configurations. Such an extension would enable the higher-level enterprise to react to certain system behaviour, and dynamically alter the lower levels. The purpose of this would be to improve performance or security of the overall solution, by manipulating sub-system implementations.

9.   References

[Carignani 2000] A. Carignani and M. De Marco. *Supporting Multiple Channel Architecture Design: The UML Contribution in a Virtual Banking Environment*. In: H.R. Hansen et. al. (eds). *Proceedings of ECIS 2000*. Vienna, 2000. Pages 883-888.

[Deitel 2001] H.M. Deitel et. al. *E-Business & E-Commerce – How to Program*. Upper Saddle River, NJ, 2001. Prentice Hall.

[DNS] Domain Names – Concepts and Facilities. RFC1034. http://www.faqs.org/rfcs/rfc1034.html.

[FreeBSD] http://www.freebsd.org.

[Ince 2002] D. Ince. *Developing Distributed and E-Commerce Applications*. Harlow, UK, 2002. Addison Wesley.

[McLaughlin 2000] B. McLaughlin. *Java and XML*. Sebastopol, CA, 2000. O'Reilly & Associates.

[Redmill 2001] F. Redmill and T. Anderson (eds). *Aspects of Safety Management*. London, UK, 2001. Springer.

[Schoder 2000] D. Schoder. *Risk in Electronic Commerce: It does matter, but not equally for all companies*. In: *Proceedings of ECIS 2000*. Vienna, 2000. Pages 838-843.

[Turban 2000] Turban et. al. *Electronic Commerce – A Managerial Perspective*. Upper Saddle River, NJ, 2000. Prentice Hall.

[XML] Extensible Markup Language (XML) 1.0 (Second Edition) http://www.w3.org/TR/2000/REC-xml-20001006

[XSL] XSL Transformations (XSLT) Version 1.0 http://www.w3.org/TR/xslt