ENVIRONMENTAL ACQUISITION IN MOBILE NETWORK SIMULATION

David Buchmann, Dominik Jungo, Ulrich Ultes-Nitsche
telecommunications, networks & security Research Group
Departement of Computer Science, University of Fribourg
Boulevard de Pérolles 90, 1700 Fribourg, Switzerland
{*david.buchmann,dominik.jungo,uun*}*@unifr.ch*

Keywords: Environmental Acquisition, Wireless Network Simulation, Validation by Simulation, Network Configuration Management.

Abstract: The type or class of an object is not always sufficient to decide on its runtime behaviour. In many cases, the context of the object is also important. Environmental acquisition is a concept to handle context in information models.
We explore the use of environmental acquisition for the management of computer network configuration and for network simulation. Physical location or subnet membership are part of the environment for a computer in a network. For the simulation of networks, environmental acquisition will be especially useful to model mobile devices moving from one area into an other.
The concept is illustrated using the network configuration management project Verinec.

1 INTRODUCTION

Class inheritance is an important concept for clean design, avoiding redundant definition. It is not only used in object oriented programming languages, but in other fields too. In network configuration, we can say a node *is of a certain type*. It might be a workstation or firewall appliance. There can be specialisations for distinct requirements on hard- and software, like a graphics workstation or a web developer workstation.

However, the *"is a"* hierarchy is not sufficient for all kind of information model. Many objects also have a *"is in"* hierarchy, that is, their *context* is relevant for their behaviour. Taking the context into account can improve network configuration management to a great extent. Networked devices are located *in a room*, appear *in the subnet* they are connected to and so on. The contexts define which printer to use by default, the network gateway and many other configuration items.

Under the term *environmental acquisition*, (Gil and Lorenz, 1996) present a concept to handle context. The idea of acquiring information from the containment hierarchy has been implicitly used in many places. Gil and Lorenz propose to explicitly model acquisition in the description language. In this paper, we will discuss the implications of environmental acquisition for modelling network configuration and for simulating networks containing mobile devices. The discussion is illustrated with examples how acquisition is used in the network configuration management system *Verinec*[1].

In the following subsection, we will briefly introduce the Verinec project and look at related work in the area of network management. Section 2 explains the concept of environmental acquisition in general. We apply the use of acquisition for network configuration management in Section 3. In Section 4, we explore the application of acquisition to network simulation in the context of simulating mobile devices operations. Finally, Section 5 holds the conclusions and presents some ideas for future research on the use of environmental acquisition in the context of network management and simulation.

1.1 The Verinec Project

With the Verinec project, we aim to outline a tool for designing and configuring heterogeneous networks.

[1]The Verified Network Configuration (Verinec) project is supported by the *Swiss National Science Foundation* under the grants number 200021-100645/1 and 200020-108056/1.

We try to bridge the gap between network discovery, network design and validation and network configuration automation. The core of Verinec is a central network definition in an XML language independent of specific products. The language defines specific expressions to configure each service, allowing the application to actually understand the semantics of the configuration data. This approach is less flexible when it comes to integrating new services. Integration would be easier when operating on implementation specific configuration files. What we gain, however, is the possibility to validate the correct semantics of the configuration. Additionally, we get the freedom of exchanging one implementation of a service for another without having to change the internal configuration representation.

The network is defined in a hierarchical way. On the lowest level of networks, we specify which network interfaces are connected physically. For an interface, we can really say that it *is in* a network. This allows to properly model nodes having more than one interface as belonging to several subnets. To augment semantic information, physical networks can be grouped into logical networks to model both geographical areas (office, floor, building) and logical groups (broadcast domains, subnets). These groups will come in handy later on, when we discuss how to avoid configuration redundancy.

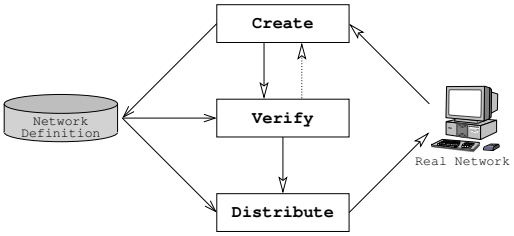

Figure 1: Verinec Workflow: Define network from existing network and user input – Validate correctness and fix mistakes – Distribute configuration.

Several modules written in Java are built around the network specification. Figure 1 depicts the complete workflow of the Verinec system.

There is a module to create and modify the XML. It has a network scanner built into it to automatically discover the layout of existing networks. Configuration importers can read configuration files of services. A graphical editor allows to correct and update the network configuration.

Having a central administration tool is convenient for administrators as they do not need to switch between tools specific to each device and it helps keeping track of the complete network inventory. More important in our context, the central configuration is a basic requirement for automated correctness testing. Once the configuration is established, we can run sanity checks on the single nodes and validate the correctness of the configuration using the *validation module*. Using simulation, we can ensure that a given configuration will not disturb network operations prior to deploying it. The validation module runs test cases to validate the network similar to unit testing in software development (Jungo et al., 2005). The simulation approach allows to find out much more than would be possible by static examination of the configuration or even by running tests on a real network. For example, we can simulate the breakdown of any device to test fail-safeness.

Having validated a configuration in a design tool does not prevent the administrator from making mistakes if he has to apply it manually to the devices. Thus, Verinec provides for automated configuration deployment to avoid introducing errors while porting the validated configuration to the actual network. The *adaption module* (Buchmann et al., 2005) translates configuration data from the XML document into a format suitable for distribution on the target device and deploys it using existing protocols for remote configuration.

1.2 Existing Work

There exist tools to specifically administer large groups of routers, e.g. by CISCO or IPAT[2]. Splat (Abrahamson et al., 2003) specialises on managing just the edge devices. Such tools are mostly used by service providers. Enterprise or university networks are a lot more heterogeneous than a service providers' infrastructure. Besides switches and routers for the internal network and external connections, one finds different servers and finally workstations that need to be configured. Verinec addresses these issues by providing a framework to add new services to configure when needed.

Several projects exist for administrating the computers in university networks. Cfengine, the "Configuration Engine" (Burgess, 1995), provides a high level scripting language and focusses on file manipulation on Unix systems. The "Local ConFiGuration system" (Anderson and Scobie, 2002) has a focus on installing software on Unix hosts. Both need an agent running on each node, contrary to Verinec. They use a less semantically rich definition language for the network than Verinec does. Netopeer (Lhotka and Novotny, 2005) uses a configuration language similar to Verinec and also uses the approach of having plugins to remotely configure the actual devices. The project is developed in the context of an IPv6 project and thus focussed on routing, while Verinec aims at configuring all network services.

[2] `www.wandl.com/html/ipat/IPAT_new.cfm`

The configuration language "Anomaly" (Logan et al., 2002) was developed to explore the possibilities of environmental acquisition for network management. It is a simple configuration language to define properties and use them in the style of logic programming. The language is less flexible to extend for new services than the Verinec XML definition. Anomaly seems to be targeted at Unix hosts only, while a central concept for the Verinec project is device independence. Work on the anomaly project seems to have ceased.

2 ENVIRONMENTAL ACQUISITION

Environmental acquisition is the process of acquiring information from the current container of an object. The concept has first been presented under this name in (Gil and Lorenz, 1996). We will use it in the context of network configuration and simulation, but the concept can be applied to all domains having objects and containment hierarchies.

Object oriented systems use the notion of "class" to form hierarchies of types. A subclass *is a* kind of its superclass. It thus has to inherit all features from the superclass, possibly extending them. The features are all known before the classes are instantiated and do not change during execution. Instances of classes have the features of their class and superclasses.

The type hierarchy is however not the only important relation an object can have. Some features depend on the containment structure. Let us look at an example. When designing a mobile phone, the manufacturer might offer differently coloured variants of the same model. The device consists of electronics, a display and a case. Display and insides will not change just because we want the colour to be different. It is the outside which is interesting here. The casing can be further divided into front side, backside and the keys.

The schematic containment hierarchy is shown in Figure 2. The parts of the casing will have a common base class, telling the material to use and similar information. For one variant of the device, the keys should have the same colour as the casing front. If we would use inheritance to ensure this, we would have to pretend that each key *is a kind of* casing front. Clearly, considering the front as a container for the keys and letting the keys acquire their colour from the container is more natural.

Acquisition allows for environmental polymorphism. Objects of the same class can show different variants of behaviour depending on their environment. This is not to be confused with subtype polymorphism, where code written to operate with a class

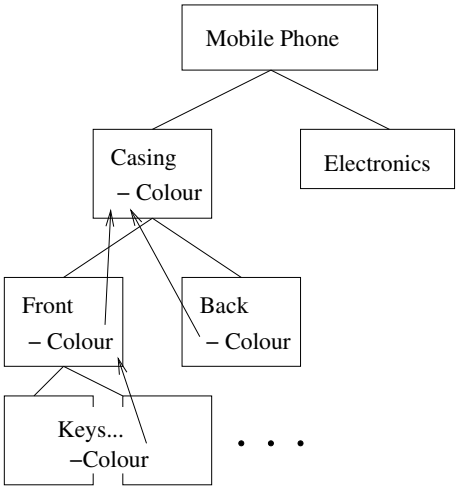

Figure 2: Containment hierarchy of a mobile phone.

is also able to handle all of its subtypes. Here, the code is parametrised by the actual object it operates on, while with environmental acquisition, an object is parametrised by its container.

In opposition to subclassing, a contained object should not acquire all features from its container. For a proper design, (Gil and Lorenz, 1996) propose to explicitly list the features to acquire from the container. To avoid long lists of features, it is possible to form logical groups and simply declare the whole group to be acquired. In the mobile phone example, this could mean that the keys acquire material surface properties like colour or roughness from the body. The shape of the part or information about the quantity of raw materials needed to build that part on the other hand are typical type information which should not be acquired but inherited.

3 ACQUISITION IN NETWORK CONFIGURATION

In large networks, many devices fulfil similar tasks. Redundancy in configuration data is a problem for network setup, as there is a considerable risk of forgetting to update mistakes in every occurrence. In larger settings, it is practically impossible to administrate all aspects of each device manually.

Templates are used to encapsulate common settings of different device types. Typically, we would use different templates for client machines, network routers, server machines and so on. This is a class hierarchy, each node *is a* certain type of machine.

However, there is information not defined by the type but by the environment of the node. For nodes in

the same subnet, many features will be or even must be the same. The most obvious is the subnet mask, also having an impact on valid static IP addresses. On the services level, think of the file server a client has to use or the name of the default printer. It is of course possible to duplicate this information in every node configuration, but when something changes, the administrator has to update every occurrence of the setting. This is not just a waste of time but also a severe threat to the reliability of a network. It is just too easy to forget to correct every occurrence of redundant information.

We could start to create special subtypes (templates) for each subnetwork. However, this is not a clean design and its limits show when we have nodes of fundamentally different types in the same subnet. If a workgroup server and client machines share a common subnet, they need to use the same gateway for accessing the rest of the network. But this does not make them be of the same type. One approach could be to use multiple inheritance, allowing a node to be of type "client" and of type "in-subnet A". However, being in a subnet is a typical containment hierarchy and is not well represented by a *is a* relationship.

This is where environmental acquisition comes in. The hierarchical network structure, as we have it available in the Verinec project, can be used to define common properties for all nodes within an area. From physical location in an office, it is for example possible to choose the correct default printer and from the subnet of a node defines network mask and default gateway.

Variable declarations are used to represent and reference context information. Instead of a concrete value, the node configuration can reference a variable. Even more interesting, the default configuration attributed to the type definition can also use the variables. The variables are evaluated in the context of the nodes of that type. A node might be completely configured by specifying its name, its type to choose the right configuration template and the context it is in.

The nodes in Verinec are considered as composites of hardware and the services they offer. It is actually the individual services which need to get properties from the network. The interfaces are bound to exactly one network and can acquire information from it. For the services of a node, we need to specify precedence to choose from which interface's environment to get information first.

3.1 An Example Network

Figure 3 shows the different relations of nodes with their context and with their classes. The machines `diufpc01` and `diufpc02` are of type client, which defines many properties like routing setup or user accounts. However, the information about the default gateway, default printer and the netmask are taken from the network of office 404, their environment. The server `diuf-serv` is of a completely different type, but gets gateway, printer and netmask from the same environment. The `d-gateway` finally is not completely in the office 404 network. Its inbound interface still acquires the netmask, but the rest is to be treated differently. The definition of interface precedence for acquisition is important here, otherwise the gateway might get assigned the default gateway of the office 404 network, which it is itself.

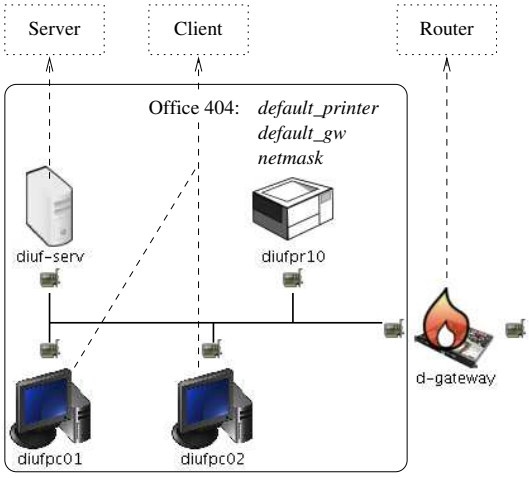

Figure 3: Network configuration with environmental acquisition.

The gateway is a good example, why acquisition should never occur implicit, but be declared explicitly. Assume the acquisition would happen automatically. The gateway would get the configuration for a default printer, which is nonsense. At the same time, the printer would be assigned a default gateway, which is not applicable neither.

If Verinec is to manage a DHCP server, it should configure it to tell the DNS server the IP's as it assigns them to clients. As long as all DHCP addresses are assigned using a fixed mapping (i.e. based on MAC addresses), DHCP and DNS servers could also share a common table.

Addresses configured statically on the interfaces provide an interesting case for the DNS server. It is redundant to define the name to IP mappings within the DNS server again. However, the DNS server is not contained in each of the nodes, but would need to acquire the mapping from the nodes. In this situation, the object needs information from its surrounding objects.

To resolve this situation, variable values can be more than simple strings, but full XML structures.

And the variable reference can specify an XPath[3] to choose only a part of a value. With this solution, the environment can define a list of mappings between host names and their static IP's. The DNS server can use the complete list for his database, and every node can select its own IP from the list using XPath.

Validating the configuration (for example using the Verinec simulator) is only a last resort to detect errors before the configuration is applied. Avoiding the mistakes being introduced into the configuration in the first place is even better. Environmental acquisition helps towards this goal by avoiding redundant information and thus the potential for conflicting settings. There are still circumstances where we need validation though, which we will discuss in the next section.

4 SIMULATING MOBILE NETWORKS

Simulation takes the central role in testing the validity of the configurations within Verinec. In this section we will introduce the basic idea of validation by simulation and discuss the impact of environmental acquisition on the simulation. Our network simulator (Jungo et al., 2004) is is based on the event based simulation framework DesmoJ (Page et al., 2000). The simulator reads the network configuration and an XML document defining a list of input events that need to be scheduled, for example the launch of an application or sending an IP packet.

The output of the simulator is an event log in the same format as the input document. Each XML element of the input document is copied to the output and the events caused by it are represented by child elements. For example, launching a `ping` command produces a series of events like MAC resolution or sending and receiving of IP packets. All those events are grouped under the "ping" input event.

The log file produced by the simulator represents the network's behaviour under the given user interaction (input events). It can be checked against a series of semantical test (Jungo et al., 2005) written in clixml (Jungo et al., 2006). These tests are similar to unit tests for programming languages and should be satisfied before the configuration is deployed to the network. The log file is enriched with meta data referencing to configuration elements involved in the behaviour that produced an event. If a semantical test fails, the meta data helps to find the configuration part, which made a test fail. After successful tests, the configuration should work as desired and be free of problematic configuration elements.

[3]XML Path Language is a compact syntax for addressing portions of an XML document.

4.1 Acquisition in the simulation

Environmental acquisition can not only be used for defining redundancy free node configuration, but also to simulate and test the configuration of a node in a dynamic environment. Introducing the concept of mobility yields the difficulty that the environment changes during the simulation.

Let us look more closely at the case of a mobile device with Wireless LAN (WLan) capabilities. The device can change its position within the network and thus move into range or out of range of base stations. For static wired networks, the network definition tells the simulator at startup on which interfaces a packet sent over a cable will arrive. Unlike wired networks, the wireless environment can only determine at runtime which packets will be received by which nodes. The mobile device needs to be in range of a base station to be able to communicate. When a device or base station sends a packet, the simulation framework needs to know which devices are in range, to decide where to schedule the packet's reception.

Wireless network client drivers often allow to be configured to connect automatically to a network. Different aspects like encryption and signal quality can be taken into account. But since our model does not know signal quality, automatic reconnection is not possible to simulate. We ignored signal quality for several seasons: For the Verinec project, we are interested in deciding whether the configuration is semantically correct. Signal quality is quite hard to predict, and simulating it would make the framework much more complicated. We thus simplify the concept by defining events which explicitly tell the simulator that a device gets into or out of range of a base station and when it initiates a connection to a network. We defined the events *get in range of*, *get out of range of*, *connect* and *disconnect*. Connect will tell the higher level services to simulate DHCP or similar protocols to establish a connection with the new network, and requires being in range of the specified base station. The listing in Figure 4 shows an extract from an event input file where first the mobile device loses connection to the base station "hall", gets into range of the base station "room" and connects to it.

```
<event time="0" node="wdevice1">
  <wlan-range type="out" base-station="hall" />
</event>
<event time="1" node="wdevice1">
  <wlan-range type="in" base-station="room" />
</event>
<event time="1" node="wdevice1">
  <wlan-connect base-station="room" />
</event>
```

Figure 4: Input events to change the environment.

From the device's point of view, changing its position means getting into a new environment. For the other devices, like mobile devices in an ad hoc network or the base station in an infrastructure mode network, the mobile device moving around changes their environment. Whenever a packet is sent over a wireless network interface, the environment of the interface is consulted to decide which interfaces are currently in range to receive the packet.

This allows to test the configuration of the services involved in handling mobile devices. As an example, for employees connecting their notebook from outside the network using a VPN[4], security policy might require that they can not access a file server with confidential files. However, if the same employee takes his notebook to the office, he must be able to use all services. This example also shows that in configuration tests, mobile does not automatically imply wireless. The concept can also be used with devices connected to different networks by cables.

Mobility has not only an impact on the simulation framework, but also on the mobile device's run-time configuration. When changing from one base station to another, the mobile device might also change its IP address and default gateway, which can be static or received dynamically via DHCP. Once the mobile device has an IP address, its behaviour in the new environment can be tested using the appropriate test cases with input events and expected outcome. However, this dynamic configuration has to be achieved by simulating network protocols. As the final aim is to have a correctly working real network, we need a realistic simulation. Environmental acquisition can not be used directly to update the mobile device's configuration.

5 CONCLUSION

In this paper, we presented the concept of environmental acquisition proposed in (Gil and Lorenz, 1996). We then explored the application of this concept to network management and network simulation.

The network defines a natural containment structure to acquire information from. Additional containments not directly visible in the network topology, like rooms or floors, can be taken into account too. The concept is quite powerful in order to reduce redundancy and thus ensure consistency of configuration data. At the same time, the effort to change the configuration is diminished. Even changing a setting that affects all workstations like the name of the file server can be done in one place and all stations are updated accordingly.

[4]Virtual Private Network is a standard to tunnel a connection over an untrusted network into the local network.

For simulation, environmental acquisition helps to model mobile devices moving in and out of reach of other wireless stations. Because we are interested in the correctness of configuration and not in the details of wireless protocols, acquisition allows us to simulate exactly the level of detail we need.

REFERENCES

Abrahamson, C., Blodgett, M., Kunen, A., Mueller, N., and Parter, D. (2003). Splat: A network switch/port configuration management tool. In *Seventeenth Systems Administration Conference (LISA 03)*, Berkeley, CA, USA. Usenix.

Anderson, P. and Scobie, A. (2002). LCFG - the Next Generation. In *UKUUG Winter Conference*. UKUUG.

Buchmann, D., Jungo, D., and Ultes-Nitsche, U. (2005). Automated configuration distribution in Verinec. In *Proceedings of the 2nd International Conference on e-Business and Telecommunications Networks (ICETE 2005)*, Reading, UK. INSTICC Press.

Burgess, M. (1995). Cfengine: a site configuration engine. In *USENIX Computing systems*, volume 8, University of Oslo, Norway. Usenix.

Gil, J. and Lorenz, D. H. (1996). Environmental Acquisition – A New Inheritance-Like Abstraction Mechanism. In *Proceedings of the 11^{th} Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 214–231. OOPSLA'96.

Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2004). The role of simulation in a network configuration engineering approach. In *ICICT 2004, Multimedia Services and Underlying Network Infrastructure*, Cairo, Egypt. Information Technology Institute.

Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2005). A unit testing framework for network configurations. In *Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification, and Validation of Enterprise Information Systems (MSVVEIS 2005)*, Miami, Florida, USA. INSTICC Press.

Jungo, D., Buchmann, D., and Ultes-Nitsche, U. (2006). Testing of semantic properties in XML documents. In *Proceedings of the 4rd International Workshop on Modelling, Simulation, Verification, and Validation of Enterprise Information Systems (MSVVEIS 2006)*, Paphos, Cyprus. INSTICC Press.

Lhotka, L. and Novotny, J. (2002-2005). Netopeer. Technical report, Cesnet, `http://www.liberouter.org/netopeer/`.

Logan, M., Felleisen, M., and Blank-Edelman, D. (2002). Environmental Acquisition in Network Management. In *Sixteenth Systems Administration Conference (LISA 02)*, pages 175–184, Berkeley, CA, USA.

Page, B., Lechler, T., and Claassen, S. (2000). *Objektorientierte Simulation in Java mit dem Framework DESMO-J*. BoD GmbH, Norderstedt.