Management of the Internet and Complex Services

European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D9.2
D9.2: Next Generation Management Technologies and Approaches to Support Autonomic Management

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
International University Bremen, IUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universidat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, UniBwM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois — Campus scientifique
615, rue de Jardin Botanique — B.P. 101
F—54600 Villers Les Nancy Cedex
France

Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title: Next Generation Management Technologies and Approaches to Support Autonomic Management

Type: Public

Editor(s): George Pavlou, Antonis Hadjiantonis

E-mail: g.pavlou@surrey.ac.uk, a.hadjiantonis@surrey.ac.uk

Author(s): Remi Badonnel, Mark Burgess, Oscar Fredy Duque Gonzalez, Antonis Hadjiantonis, Iris Hochstatter, Ralf König, Emil Lupu, Apostolos Malatras, Krzysztof Nowak, George Pavlou, Aiko Pras, Javier Rubio-Loyola, Jürgen Schönwälder , Joan Serrat, Rolf Stadler, Tarun Varma (in alphabetical order)

Doc ID: D9.2-v1.0.pdf

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1	2007-02-21	Antonis Hadjiantonis	First version, providing the ToC
V0.2	2007-05-21	See authors list	Initial draft for internal commenting
V0.2.1	2007-06-08	See authors list	First draft for partner commenting
V0.3	2007-06-22	See authors list	Partner comments addressed
V0.4	2007-07-04	Antonis Hadjiantonis, George Pavlou	Final Editing
V1.0	2007-07-04	Antonis Hadjiantonis	Final Version

Legal Notices

Table of Contents

(This page is left blank intentionally)

Executive Summary

The concept of Autonomic Management is receiving intense interest from both academia and industry since it emerges as an appealing solution to the increasing complexity of managing IT systems. This document provides a detailed view of today's **Next Generation Management Technologies and Approaches to Support Autonomic Management**. It is the outcome of ongoing research integration and collaboration among partners of EMANICS Network of Excellence and in particular Work Package 9 (WP9: Autonomic Management).

Having investigated Frameworks and Approaches for Autonomic Management of Fixed QoS-enabled and Ad Hoc Networks, those results were published in Deliverable D9.1. Building on that knowledge, this deliverable provides a concrete view of technologies and approaches that can realise Autonomic Management. This realisation is at its infancy and the presented ongoing research efforts are only the first steps to tackle such a colossal task. However, tangible benefits like self-configuration, self-healing and self-optimization fuel interest and motivate our work.

The **policy-based management (PBM) paradigm** is often linked directly to autonomic management. The specification of high level objectives and their automated deployment and lifecycle management within complex networks make PBM approaches and technologies important candidates for Autonomic Management realisation. In Chapter 2 we explore the critical aspects of policy analysis, having identified the importance of consistency and automated conflict resolution for policy-based systems.

Autonomic Management can benefit from **Web Services-based approaches** by exploiting their unique features. Self-configuration can be significantly simplified if build on a platform independent and interoperable framework based on Web Services. In addition, self-awareness and self-optimisations can be catered by a WS-based monitoring and notifications framework. In Chapter 3 we present an overview of Web Services technology and identify how Autonomic Management can benefit. Technologies based on XML, including Web Services, can provide a rich communications framework that goes over the limitations of traditional management protocols and is open to a large developer community.

An autonomic system can be viewed as a system capable of managing itself and adapting to changes in accordance with policies and objectives defined by a human administrator. This system can interact with other autonomic systems to perform collaborative tasks. In this case, trust is a crucial issue to evaluate how an autonomic system (or an autonomic component) is reliable and if it will interact properly when it is solicited by another one. Therefore in Chapter 4 we investigate issues of **trust and misbehaviour** in Autonomic Systems.

We present recent efforts towards **context awareness** since this is a critical requirement for autonomic systems. Chapter 5 follows a thorough investigation of context awareness in D9.1 and presents here novel efforts and techniques. Finally, Chapter 6 provides an overview of **inspiration domains** for systems with self-management capabilities and outlines potential directions and ideas for our future work.

1 Introduction

In an increasing complex world of IT and Telco systems there is an attractive opportunity for Autonomic Management concepts to provide solutions. In this document we provide a detailed view of Next Generation Management technologies and approaches to support Autonomic Management. It is the outcome of ongoing research integration and collaboration among partners of the EMANICS Network of Excellence and in particular Work Package 9 (WP9: Autonomic Management).

The concept of Autonomic Management [1,2,3] is receiving intense interest from both academia and industry since it emerges as an appealing solution to the increasing complexity of managing IT systems. Having investigated Frameworks and Approaches for Autonomic Management of Fixed QoS-enabled and Ad Hoc Networks in Deliverable D9.1, we build on that knowledge. This deliverable provides a concrete view of technologies and approaches that can realise Autonomic Management. This realisation is at its infancy and the presented ongoing research efforts are only the first steps to tackle such a colossal task. However, tangible benefits like self-configuration, self-healing and self-optimization fuel interest and motivate our work.

Policy based approaches to network and systems management are of particular importance because they allow the separation of the rules that govern the behaviour of systems from the functionality provided by that system. This means that it is possible to adapt the behaviour of a system without the need to recode functionality, and changes can be applied without stopping the system. This provides autonomic behaviour in the form of an efficient closed feed-back loop either locally or hierarchically. Research into policy-based systems management has focussed on languages for specifying policies and architectures for managing and deploying policies in distributed environments. However, policy based techniques are not in widespread use because their advantages in terms of short term return on investment are difficult to justify without significant advances on tools for policy analysis and refinement. Much of the value of policy-based techniques derives from being able to analyse a policy specification in order to detect conflicts and validate that it can be enforced on the available resources. The ability to analyse a set of policies and offer guarantees regarding their consistency and enforcement is one of the primary benefits of restricting policy languages to declarative specifications and of investing in policy techniques thus significantly reducing the costs associated with maintenance and re-configuration in autonomic networking. Despite these advantages policy analysis presents significant research challenges. In Chapter 2 we explore the critical aspects of policy analysis, having identified the importance of consistency and automated conflict resolution for policy-based systems.

An important technology to consider for the realisation of Autonomic Management solutions are Web Services (WS) and XML related approaches. Although these technologies have been used for years, recent technological advances and optimisation efforts have made them very attractive for managing complex networks. Considering the fact that the management of networks and systems still relies on protocols and technologies that were developed several decades ago, like CLI and SNMP, the new emerging paradigm of Web Services-based Management is gaining pace. In this sense, Web Services can be an appealing enabler of Autonomic Management. Autonomic Management can benefit from Web Services-based approach by exploiting their unique features. Self-configuration can be significantly simplified if build on a platform independent and interoperable framework based on Web Services. In addition, self-awareness and self-optimisations can be catered by a WS-based monitoring and notifications framework. In Chapter 3 we present an overview of Web Services technology and identify how Autonomic Management can benefit. Technologies based on XML, including Web Services, can provide a rich communications framework that goes over the limitations of traditional management protocols and is open to a large developer community.

The deployment of trust and reputation mechanisms in autonomic systems requires that the systems observe one another in order to determine to what extent they behave in accordance with what they have implicitly or explicitly claimed to the others (their “promises”). In addition an observing system can, in turn, be evaluated by the other systems to determine the reliability of its observations. As information about trust evaluations are made public or spread throughout a collaborative group, one can talk about the concept of reputation, or trust within a community. In Chapter 4 we investigate issues of trust and misbehaviour in Autonomic Systems. This document will detail key concepts of trust in autonomic systems. First, we will present how the autonomic systems can describe themselves and advertise their behavioural properties. Then, we will show how these properties can be monitored in an autonomic manner and to what extend analysis models are necessary to improve the efficiency of this monitoring. A third section will discuss how these autonomic systems can estimate, based on the observations, whether a system is reliable or not. Finally, we will show how trust is a matter of policy and how it relates to reputation.

Communication networks nowadays tend to become a common utility, used by more and more people on everyday basis in an increasingly complex way. User may expect its terminal device to perform differently depending on various properties of the surroundings, adapting itself to the varying conditions and providing the user with best possible experience under current circumstances. In order to meet such requirements, both end-user devices and the network must be capable of gathering and processing information describing the operational environment and enabling the services to make use of such information. This kind of information, a “context”, may be useful in dynamic configuration of network parameters, as defined in autonomic management paradigm, but it may also be of much use in case of high-level applications, such as, for instance, travel route planning for mobile subscribers, gas station finders, weather forecast services etc.. This creates a need for a translation layer that would enable different context providers to work with various types of devices. We present recent efforts towards context awareness since this is a critical requirement for autonomic systems. Chapter 5 follows a thorough investigation of context awareness in D9.1 and presents recent research efforts.

Finally, Chapter 6 provides an overview of inspiration domains for systems with self-management capabilities and outlines potential directions and ideas for our future work. This chapter argues that we can expedite progress in systems with self-management capabilities (management systems, assistance systems, support systems) in IT management by getting inspiration by existing knowledge in IT management itself or other domains. After introducing a model for the knowledge in and across domains it classifies ways of inspiration and lists promising inspiration domains.

1.1 Purpose of the Document

The purpose of this document is to provide a detailed view of today's Next Generation Management Technologies and Approaches to Support Autonomic Management. It is the outcome of ongoing research integration and collaboration among partners of the EMANICS Network of Excellence and in particular Work Package 9 (WP9: Autonomic Management).

Having investigated Frameworks and Approaches for Autonomic Management of Fixed QoS-enabled and Ad Hoc Networks in Deliverable D9.1 we build on that knowledge to provide a concrete view of technologies and approaches that can realise Autonomic Management. This realisation is at its infancy and the presented ongoing research efforts are only the first steps to tackle such a colossal task. However, tangible benefits like self-configuration, self-healing and self-optimization fuel interest and motivate our work.

1.2 Document Outline

The document is organized as follows. After an executive summary of this deliverable, **Chapter 1** serves as a general introduction, while it also presents the purpose of this deliverable and its outline. In **Chapter 2** we explore the critical aspects of policy analysis, having identified the potential of policy-based management approaches and the importance of policy consistency. The specification of high level objectives and their automated deployment and lifecycle management within the network make the PBM approach and technologies important candidates for Autonomic Management realisation. In **Chapter 3** we present an overview of Web Services technology and identify how Autonomic Management can benefit. Technologies based on XML, including Web Services, can provide a rich communications framework that goes over the limitations of traditional management protocols and is open to a large developer community. In **Chapter 4** we investigate issues of trust and misbehaviour in Autonomic Systems. Trust is a crucial issue to evaluate how an autonomic system (or an autonomic component) is reliable and if it will interact properly when it is solicited by another one. In **Chapter 5** we present recent efforts towards context awareness since this is a critical requirement for autonomic systems. Finally, **Chapter 6** provides an overview of inspiration domains for systems with self-management capabilities and outlines potential directions and ideas for our future work.

2 Policy-based paradigm for autonomic management

Policy based approaches to network and systems management are of particular importance because they allow the separation of the rules that govern the behaviour of systems from the functionality provided by that system. This means that it is possible to adapt the behaviour of a system without the need to recode functionality, and changes can be applied without stopping the system. This provides autonomic behaviour in the form of an efficient closed feed-back loop either locally or hierarchically. Research into policy-based systems management has focussed on languages for specifying policies and architectures for managing and deploying policies in distributed environments. However, policy based techniques are not in widespread use because their advantages in terms of short term return on investment are difficult to justify without significant advances on tools for policy analysis and refinement. In effect, many vendors and system administrators tend to use existing scripting or general purpose programming languages in order to program configuration changes. The benefits of using more restrictive declarative policy languages such as Ponder, PDL, or CIM-SPL arise only if the policy specification can be analysed for conflicts and inconsistencies, if it is possible to verify that a policy specification preserves well-defined properties and if tools for refining policies from high-level business objectives and Service Level Agreements are available.

Much of the value of policy-based techniques for network and systems management derives from being able to analyse a policy specification in order to detect conflicts and validate that it can be enforced on the available resources. The ability to analyse a set of policies and offer guarantees regarding their consistency and enforcement is one of the primary benefits of restricting policy languages to declarative specifications and of investing in policy techniques thus significantly reducing the costs associated with maintenance and re-configuration in autonomic networking. Despite these advantages policy analysis presents significant research challenges. Whilst early work [Lup99] has demonstrated the feasibility of detecting a number of potential conflicts through syntactic analysis, this does not account for the fact that policy applicability is restricted through constraints and thus conflicting policies may not apply simultaneously. Policy constraints limit the applicability of policies and may be expressed in terms of time, context or state of managed objects including resource usage. In order to be able to analyse a set of policies whilst taking into account their applicability it is necessary to use formal techniques. This, in turn, requires a formalisation of the policy enforcement as well as a model of the system behaviour. In this section we present an analysis technique that provides the means to analyse a policy specification in terms of detecting inconsistencies and ensuring that policies exhibit pre-defined properties. This technique can be used in conjunction with and simultaneously with refinement techniques ensuring policy consistency and properties at each of the refinement levels in a refinement hierarchy. We will first present the policy formalisation (i.e., formal model and semantics for policy enforcement) for the Ponder language [4] and then the policy analysis capabilities. An alternative approach is also presented based on Finite State Transducers (FST) concepts. This proposal is an approach consisting of an adapted subset of finite state automata concepts, to detect and solve conflicting policy rules. They are special automata for which, on each edge, there are two labels instead of one. A model based on finite state transducers is not a completely new idea, but we think that its full potential has not been exploited. In particular our transducers will not model the network behaviour, but rather the policies themselves.

Self-configuration is a key property of Autonomic Systems and is widely accepted that policy-based management can address its requirements effectively. Towards that direction, a policy refinement methodology adds an extra layer of automation by simplifying the process of creating enforceable low-level policies. In addition, the system is able to select the optimal method in terms of selected policies, in order to achieve the defined high-level goals. This procedure is in essence Self-Optimisation of the system, as the best available combination of policies is enforced without additional human intervention.

2.1 Policy Formalisation

Existing research on policy-based systems has identified several types of policy that are useful in managing distributed systems [4]. Broadly, policies can by classified into authorisation policies and management policies where the former category captures the access control requirements of a system and the latter category holds requirements related to the system behaviour. The Ponder language, developed at Imperial College, is a declarative language that supports both of these policy types.

Authorisation policies specify whether a subject is permitted perform a particular action on a target. In a closed system, with a default policy of prohibiting all subjects from performing operations on all targets, positive authorisation policies would be used to explicitly specify which particular operations a subject is permitted to perform on a target. Alternatively, in an open system, where by default all operations are permitted, negative authorisations would be used to specify that a subject is not permitted to perform an operation on a target. A policy-based access control system is the combination of the policies that specify the permitted/prohibited operations, an access control model that defines how the permissions are organised across the system, and a reference monitor that uses the access control model to enforce the policies.

Obligation policies specify management operations that must be performed when a particular event occurs given some supplementary conditions being true. They are specified in terms of a subject that should perform a particular action on a target when a specified condition is true. Obligation policies are event based and therefore the occurrence of the specified event is a necessary condition for the mandated operation to be performed. Another difference is that obligation policies cause the agent enforcing the policy to actually perform the specified action rather than just specify that the operation is permitted.

Refrain policies allow the administrator to specify conditions under which certain operations should not be performed. They are similar to negative authorisation policies as they are both used to prevent an action from being performed on a target. However, unlike authorisations, which are interpreted by the target object's access controller, refrain policies are interpreted by the subject and can be used in situations where the target does not wish to be protected from the subject such as information disclosure policies.

Prior work on policy specification has illustrated the power of using a domain model as a tool for organising objects in a system. Domains provide a means of grouping objects to which policies apply and can be used to partition the objects in large systems according to geographical boundaries, object type, responsibility and authority. Membership of a domain is explicit and not defined in terms of a predicate on object attributes. An advantage of specifying policy scope in terms of domains is that objects can be added and removed from the domains to which policies apply without having to change the policies [5].

The Ponder language provides support for specifying authorisation, obligation and refrain policies. Its object oriented features and grouping constructs facilitate ease of use and scalability to large systems and large numbers of policies. However, Ponder is not a logic-based language and does not provide direct support for formal reasoning methods or for expressing general models of system behaviour. Therefore, Ponder cannot account for the effect of policies on system state and cannot be used directly for policy analysis. However, as will be shown here, it is possible to transform Ponder policies into a formal representation that supports both a description of the system behaviour and formal reasoning techniques for policy analysis.

Event Calculus (EC) is a formal language for representing and reasoning about dynamic systems. Because the language supports a representation of time that is independent of any events that might occur in the system, it is a particularly useful way to specify a variety of event-driven systems. Since its initial presentation [6], a number of variations of the Event Calculus have been presented in the literature [7]. In this work we use the form presented in [8], consisting of (i) a set of time points (that can be mapped to the non-negative integers); (ii) a set of properties that can vary over the lifetime of the system, called fluents; and (iii) a set of event types. In addition the language includes a number of base predicates, initiates, terminates, holdsAt, happens, which are used to define some auxiliary predicates; and domain independent axioms. These are summarised below:

Base predicates:

initiates(A,B,T)	event A initiates fluent B for all time > T.
terminates(A,B,T)	event A terminates fluent B for all time > T.
happens(A,T)	event A happens at time point T
holdsAt(B,T)	fluent B holds at time point T. This predicate is useful for defining static rules (state constraints).
initiallyTrue(B)	fluent B is initially true.
initiallyFalse(B)	fluent B is initially false.

Auxillary predicates:

clipped(T1,B,T2)	fluent B is terminated sometime between timepoint T1 and T2.
declipped(T1,B,T2)	fluent B is initiated sometime between timepoint T1 and T2.

Domain independent axioms:

holdsAt(B, T1) ←	holdsAt(B, T) ∧ ¬ clipped(T, B, T1) ∧ T<T1.
holdsAt(B, T1) ←	initiates(A, B, T) ∧ happens(A, T) ∧ ¬ clipped(T, B, T1) ∧ T<T1.
¬holdsAt(B, T1) ←	¬holdsAt(B, T) ∧ ¬ declipped(T, B, T1) ∧T<T1.
¬holdsAt(B, T1) ←	terminates(A, B, T) ∧ happens(A, T) ∧ ¬ declipped(T, B, T1) ∧ T<T1.

This is the classical form of the Event Calculus where theories are written using Horn clauses. The frame problem is solved by circumscription, which allows the completion of the predicates initiates, terminates and happens, leaving open the predicates holdsAt, initiallyTrue and initiallyFalse. This approach allows the representation of partial domain knowledge (e.g. the initial state of the system). Formulae derived by the Event Calculus are in effect classically derived from the circumscription of the EC representation. To provide an implementation of such a Calculus in Prolog, we use pos and neg functors. The semantics of the Prolog implementation assumes the Close Word Assumption (CWA) and models are essentially Herbrand models where predicates are appropriately completed. The use of pos and neg functions on the fluents allows us to keep open the interpretation of fluents being true/false, in the same way as circumscription does in the classical representation. In this way we can guarantee that the implementation of our EC is sound and complete with respect to the classical EC formalisation. The correspondence between the classical EC with circumscription and the logic program implementation can be found in [7].

The Event Calculus supports deductive, inductive and abductive reasoning. Deduction uses the description of the system behaviour together with the history of events occurring in the system to derive the fluents that will hold at a particular point in time. Induction aims to derive the descriptions of the system behaviour from a given event history and information about the fluents that hold at different points of time. However, the reasoning technique that is of particular interest to our work is abduction. Given the descriptions of the behaviour of the system, abduction can be used to determine the sequence of events that need to occur such that a given set of fluents will hold at a specified point in time.

The work described in [8] outlines how abduction can be used in conjunction with Event Calculus to analyse requirements specifications and presents a specialised set of Event Calculus axioms that reduce the computational complexity of the abductive proof procedure.

Because the enforcement of an obligation policy will change the state of the system, in addition to modelling the policy specification, it is necessary to model the system itself when developing a formal technique for analysing policies. To achieve a complete specification that supports formal reasoning, the following domain-specific information must be represented in the model.

1. Objects and their organisation into domains.
2. Available management operations and their effect on the managed objects.
3. Policy rules.

Additionally, it is also necessary to define domain independent rules for modelling policy enforcement. In order to support the transformation of this information from high-level representations into a logical notation, we use the following constants, variables, functions and predicates:

- *Constant Symbols:* Every member of Obj, where Obj represents the set of objects in the system.
- *Variable Symbols:* These are defined using the set, VO, representing the attributes of objects and VP, representing the set of parameters for the operations supported by the objects.
- *Function Symbols:* The language supports a number of functions that can be used as parameters in the basic predicate symbols of Event Calculus (Table 2-1)
- *Predicate Symbols:* In addition to the previously described Event Calculus predicates, initiates, terminates, happens, holdsAt and initiallyTrue, the language includes the predicate symbols defined in Table 2-2.

Function Symbol	Description
state(Obj, VO, Value)	Represents the value of a variable of an object in the system. It can be used in an initiallyTrue predicate to specify the initial state of the system and also as part of rules that define the effect of actions.
operation(Obj, Action(VP))	Used to denote the operations specified in a policy function or event (see below)
systemEvent(Event)	Represents any event that is generated by the system at runtime and is used to trigger enforcement of obligation or refrain policies. The Event argument specified in this term can be any application specific predicate or function symbol.
doAction(ObjSubj, operation(ObjTarg, Action(VP)))	Represents the event of the action specified in the operation term being performed by the subject, ObjSubj, on the target object, ObjTarg.
requestAction(ObjSubj, operation(ObjTarg,Action(VP)))	Represents the event that occurs whenever a subject attempts to perform an operation on a target object. Therefore, this is the event that will trigger a permission (or denial) decision to be taken by the target object's access controller.
rejectAction(ObjSubj, operation(ObjTarg,Action(VP)))	Event that occurs after the enforcement decision to reject the request by a particular subject to perform an action is taken.
permit(ObjSubj, operation(ObjTarg, Action (VP)))	Represents the permission granted to a subject, ObjSubj, to perform the action defined in the operation on the target, ObjTarg.
deny(ObjSubj, operation(ObjTarg, Action (VP)))	Used to denote that the subject, ObjSubj, is denied permission to perform that action on the target, ObjTarg.
oblig(ObjSubj, operation(ObjTarg, Action (VP)))	Denotes that the subject, ObjSubj, should perform the action specified in the operation term on the target, ObjTarg.
refrain(ObjSubj, operation(ObjTarg, Action (VP)))	Denotes that the subject, ObjSubj, should not perform the action specified in the operation term on the target, ObjTarg.

Table 2-1: Function Symbols.

Symbol	Description
object(Obj)	Used to specify that Obj is an object in the system.
attr(Obj, V_O)	Specifies that Vo is an attribute of the object, Obj.
method(Obj, Action(V_P))	Represents an action supported by an object in the system. It will be used to define a separate ground term for every operation specified in the system.
isDomain(Obj)	Defines that Obj represents a domain. In order to indicate that a domain is a specialisation of an object, we also define the following rule: object(Obj) $\leftarrow$ isDomain(Obj).
isMember(Obj, Dom)	Holds if the object, Obj, is a member of the Domain, Dom.
isSubDomain(Dom1, Dom2) $\leftarrow$ isDomain(Dom1), isDomain(Dom2), isMember(Dom1, Dom2), Dom1 != Dom2, ¬ isSubDomain(Dom2, Dom1).	Holds if the domain represented by Dom1 is a sub-domain of Dom2. The body of the rule is used to ensure that there are no cyclic relationships in the domain structure.

…table continued

Symbol	Description
isDerivedMember(Obj, Dom) ← object(Obj), ¬ isDomain(Obj), isMember(Obj, Dom). isDerivedMember(Obj, Dom) ← object(Obj), ¬ isDomain(Obj), subDomain(Dom, SubDom), isDerivedMember(Obj, SubDom).	Used to determine membership of a domain across the entire domain structure. This first rule identifies all those objects that are direct members of the domain, Dom. The second rule recursively identifies those objects that are members of sub-domains of the domain, Dom.
isValidSpec(Obj_{Subj}, operation(Obj_{Targ}, Action(V_P)) ← object(ObjSubj), object(ObjTarg), method(ObjTarg, Action(V_P)).	Many of the function definitions above contain the tuple (Obj_{Subj}, operation(Obj_{Targ}, Action(V_P)). The isValidSpec predicate is defined to hold if the members of this tuple are consistent with the specification of the managed system. As such it is used in the body of any rule where functions with the tuple (Obj_{Subj}, operation(Obj_{Targ}, Action(V_P)) are specified in the head.

Table 2-2: Predicate Symbols

Having specified the language, it is now possible to explain how the various symbols defined above can be incorporated into rules that represent the different types of information required for modelling a managed system. The sequel presents the form of these rules and illustrates their use through a simple example.

2.1.1 Policy Enforcement Model

Analysis of policies requires the ability to determine the effect of a specified policy on the behaviour of the system. Therefore, in addition to modelling the policy specification, it is necessary to define rules that model the enforcement of the policies. Such rules have the effect of linking the policy specification to the system behaviour specification.

The complete policy enforcement model is illustrated in Figure 2-1. As shown, a system event is received by the subject's policy agent, which refers to the policy repository to determine if any of the obligation policies for this subject specify this event as a trigger.

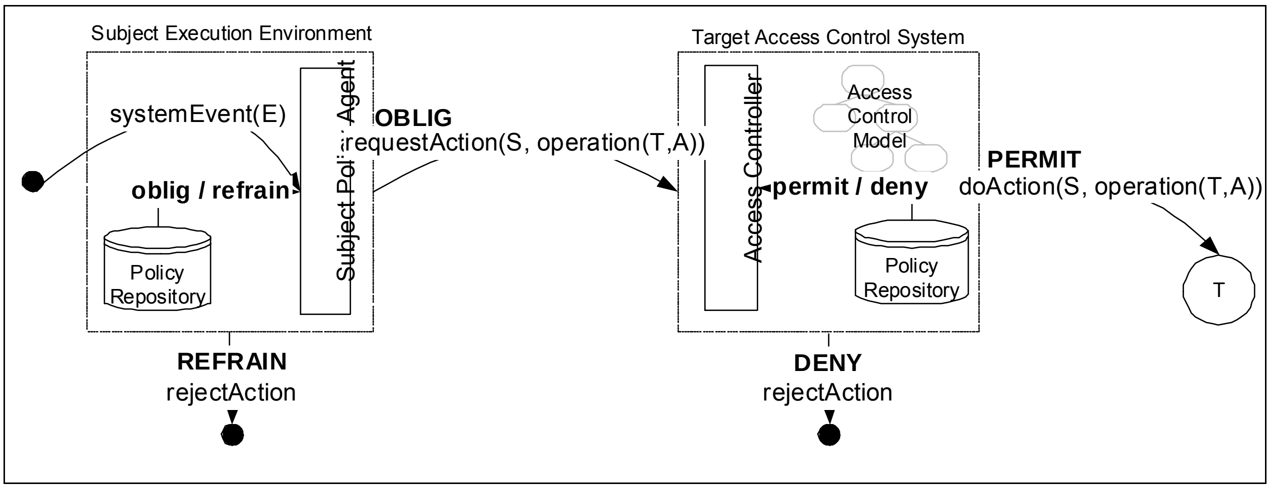


Figure 2-1: Policy Enforcement Model

If there is an obligation, this will cause a request to perform the specified action to be sent to the target. If a refrain policy that prohibits this exists at the subject, then the action will be rejected. Once the subject makes a request to perform an action on the target, the target object's access controller processes it. To do this, the access controller evaluates the request by referring to the policy repository and the access control model of the system. If the action is permitted, the access control system will proceed to do the requested action. Otherwise, if the action should be denied, the access control system will reject the action.

The formal representation of this policy enforcement model presented is presented in the text frame below. The first rule models the behaviour of subject's policy agent, causing the event of requesting an action whenever an obligation that specifies that action holds. The next rule models a subject's policy enforcement code rejecting the specified action to enforce a refrain. The third rule models the behaviour of the target's access controller, generating a *doAction* event when an action is permitted. This event would trigger the relevant system behaviour rules thus causing the system state to change according to the specification. The last rule models a target object's access controller rejecting the action to prevent a denied operation from being performed.

```
% Obligation / Refrain Enforcement Rule (Subject)
happens(requestAction(Subj, operation(Targ, Action(ParmList))), Tn) ←
 holdsAt(oblig(Subj, operation(Targ, Action(ParmList))),Tm) ∧ (Tm < Tn).

happens(rejectAction(Subj, operation(Targ, Action(ParmList))), Tn) ←
 holdsAt(refrain(Subj, operation(Targ,Action(ParmList))),Tm) ∧ (Tm < Tn).

% Access Control Rule (Target)
happens(doAction(Subj, operation(Targ, Action(ParmList))), Tn) ←
  holdsAt(permit(Subj, operation(Targ, Action(ParmList))), Tm) ∧ (Tm < Tn).

happens(rejectAction(Subj, operation(Targ, Action(ParmList))), Tn) ←
  holdsAt(deny(Subj, operation(Targ, Action(ParmList))), Tm) ∧ (Tm < Tn).
```

The final step in developing this logical notation is to represent the policies themselves. As discussed in the previous sections, we are focussing on four types of policy – positive authorisation, negative authorisation, obligation and refrain.

In order to correctly interact with the enforcement model described above, each policy specification rule should initiate the appropriate policy function symbol (permit, deny, oblig or refrain) for each of the events. So for example, a positive authorisation policy rule should specify that permit(Subj, Operation) holds when the requestAction(Subj, Operation) event occurs and the constraints that control the applicability of the policy hold. Additionally, the fluent permit(Subj, Operation) should cease to hold once the action has been performed thus making it possible to re-evaluate the policy rule on subsequent requests to perform the action. The Event Calculus representation of this functionality is shown in the (*posAuth*) specification below. We also show how each of the other policy types would be represented by rules in the formal notation.

For each rule, the terms, ObjSubj, ObjTarg, Action and Constraint, can be directly mapped to the subject, target, action, constraint and event clauses used when specifying policies in a language like Ponder. Although Ponder constraints are specified using the Object Constraint Language (OCL), typical constraints only use a subset of features from this language. As such, the Constraint predicates in the Event Calculus rules above, can be represented by a combination of holdsAt terms. Beckert et al. [9]

describe approaches for mapping general OCL specifications into first order logic. This could be used to handle more complex OCL constraint expressions. The validSpec predicate, which is the second predicate in the body of these policy rules, is used to check that the objects and operations specified in the rules are consistent with the system description.

```
(posAuth)- initiates(requestAction(Obj_Subj,operation(Obj_Targ,Action(ParmList))),
                     permit(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm) ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))) ∧ Constraint.

           terminates(doAction(Obj_Subj, operation(Obj_Targ, Action(ParmList))),
                      permit(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm) ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))).

(negAuth)- initiates(requestAction(Obj_Subj,operation(Obj_TargAction(ParmList))),
                     deny(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm)   ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))) ∧ Constraint.

           terminates(rejectAction(Obj_Subj, operation(Obj_Targ, Action(ParmList))),
                     deny(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm) ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))).
(oblig)-  initiates(systemEvent(E),
                    oblig(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm) ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))) ∧ Constraint.

          terminates(doAction(Obj_Subj, operation(Obj_Targ, Action(ParmList))),
                    oblig(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm) ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))).

(refrain)- initiates(systemEvent(_),
                    refrain(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm)   ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))) ∧ Constraint.

           terminates(rejectAction(Obj_Subj, operation(Obj_Targ, Action(ParmList))),
                    oblig(Obj_Subj, operation(Obj_Targ, Action(ParmList))), Tm) ←
              validSpec(Obj_Subj, operation(Obj_Targ, Action(ParmList))).
```

The (negAuth) rule represents a negative authorisation policy by stating that, if the Constraint holds and the event requesting the action happens, the action is denied. The second part of the (negAuth) rule shows how the deny fluent will be terminated once the decision to reject that action has been taken, thus allowing the rule to be re-evaluated on subsequent requests. Note that the termination rules for these policies do not have any constraints and can be generically specified for the whole system.

The (oblig) rule states that if the Constraint holds at the time that the system event, systemEvent(E), occurs, then the obligation for the subject to perform the action on the target holds. Like with the (posAuth) rule, we define that the obligation is terminated once the call to perform the specified operation is made. This assumes that the execution of the operation is an atomic process, i.e. the execution of the operation is considered complete once a call to the operation has been made. The (refrain) rule states that if the Constraint holds and any system event occur, the subject should not perform the action on the target because the refrain holds. Just like with the (negAuth) rule, the second part of the (refrain) rule defines that the refrain fluent is terminated once the policy enforcement decision to not perform the specified action is taken.

A complete policy specification would involve instantiating the initiates rules defined above with specific subjects, targets and operations defined for the managed system. The rules simply define the conditions under which a policy holds in the system.

2.1.2 System Behaviour Model

We now extend the language above, using Event Calculus, to model the operations supported by the system and their behaviour. This representation is automatically derived from a state-chart model of the managed objects' behaviour. The method *symbol* defined earlier is used to represent the operations that are supported by the objects in the system. In order to model the behaviour of these operations, it is necessary to specify the pre- and post-conditions for each operation. Performing an operation on the system will modify the state of the system in such a way that, once the operation is complete, there will be some new fluents that hold, and some other fluents that cease to hold. This is represented using the initiates and terminates predicates, which are defined in the Event Calculus, according to the following schema:

```
initiates(doAction(Obj_Subj, operation(Obj_Targ, Action(Parms))), PostTrue, Tm) ←
  validSpec(Obj_Subj, operation(Obj_Targ, Action(Parms))) ∧ PreCondition.

terminates(doAction(Obj_Subj, operation(Obj_Targ, Action(Parms))), PostFalse, Tm) ←
  validSpec(Obj_Subj, operation(Obj_Targ, Action(Parms))) ∧ PreCondition.
```

The first rule above states that when the doAction event occurs at time, Tm, if the PreConditions are true, then the fluent defined by PostTrue will hold after that time. Under the same conditions, the second rule states that the fluent defined by PostFalse will cease to hold after time, Tm. In both of these rules, the PreCondition will be represented by a conjunction of holdsAt terms, which are defined as part of the Event Calculus. The PostTrue and PostFalse fluents are defined using state terms that are defined in the formal language above. The validSpec predicate is used to ensure that the objects and operations specified in the rule are consistent with the specification of the objects and their organisation e.g., the action specified is an operation defined in the interface of the managed object.

2.1.3 Example

The use of the policy specification rules defined previously can be illustrated in a simple printer management example that includes a range of policy rules. Policies could be used to specify the types of process that are allowed to access the print queue. For example, only root processes are allowed to indiscriminately delete jobs from a queue. A user process is only allowed to delete a print job if it has the same process identifier as the process that originated the job. The print manager should handle an outOfPaper event by switching to an alternative input tray also reporting the event.

```
object(printer-crimson).

attr(printer-crimson, status).
method(printer-crimson, printDoc).
method(printer-crimson, switchPaper).

isDomain(office).

isDomain(bw-printers).
isDomain(highvol-printers).

isDomain(lab).

isMember(lab, office).
isMember(highvol-printers, lab).
isMember(bw-printers, lab).
isMember(printer-crimson, lab).
isMember(printer-crimson, highvol-printers).
isMember(printer-crimson, bw-printers).
```

Consider an organisation that has a number of different printers distributed through its offices. The printers are organised according to properties like the type (colour/ b&w),

capacity (high volume/ low volume) and physical location (4th floor/ 5th floor/ lab). The printers themselves are uniquely named (skyblue, violet, cobalt, grey, crimson, damson). Considering each of the properties to be represented by a different domain, the formalism presented above can be used to represent the printer crimson as an object in this domain structure.

Consider that a print manager controls every printer in the system. The print manager provides functions for viewing the printer queue, adding and deleting a print job. Additionally it is possible for the printers to provide diagnostic information (such as a paper jam) to the print manager. The print manager can use the diagnostic information to correct errors, or report the printer status to a central management console that is monitored by an administrator. A state chart representation of this functionality is shown in the figure below.

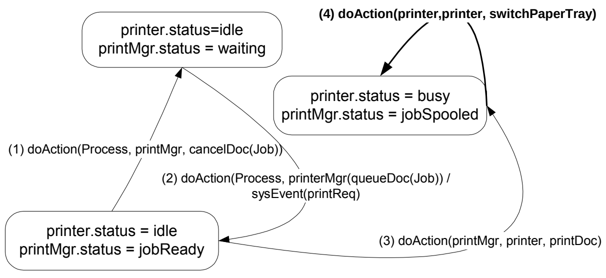

printer.status=idle
printMgr.status = waiting
(4) doAction(printer,printer, switchPaperTray)
printer.status = busy
printMgr.status = jobSpooled
(1) doAction(Process, printMgr, cancelDoc(Job))
(2) doAction(Process, printerMgr(queueDoc(Job)) / sysEvent(printReq)
printer.status = idle
printMgr.status = jobReady
(3) doAction(printMgr, printer, printDoc)

Figure 2-2: State chart for Printer System Functionality

It is possible to transform this state chart into the Event Calculus (EC) notation presented previously where the input shown on each transition arrow is the action being performed; for transition between different states, the current state values become the PostFalse fluents; any actions associated with the transition and next state values become the PostTrue fluents; and the current state values become the PreConditions. Self-transitions should not specify the current state as PostFalse fluents. So following this scheme, transition (4) in Figure 2-2 would be represented in the EC as follows:

```
initiates(doAction(printer, operation(printer, switchPaper)),
                   state(printer, status, busy), T) ←
  holdsAt(pos(state(printer, status, busy)), T) ∧
  holdsAt(pos(state(printMgr, status, jobSpooled)), T).
initiates(doAction(printer, operation(printer,switchPaper)),
                   state(printMgr,status,jobSpooled), T) ←
  holdsAt(pos(state(printer, status, busy)), T) ∧
  holdsAt(pos(state(printMgr, status, jobSpooled)), T).
```

Consider the following Ponder policies defined for the printing system:

```
// only a root process can cancel a print job
auth+  cancelJobRoot {
       subject    process/;
       target     printManager;
       action     cancelDoc(Job);
       when       process.owner == root;
}

// non root processes cannot cancel print jobs if the job being cancelled
// is not owned by the requesting process
auth-   cancelJobOther {
       subject   process/;
       target    printManager;
       action    cancelDoc(Job);
       when      process.owner != root &&
                   Job.owner != process.owner;
}
```

```
// Upon system shutdown, any jobs owned by running processes should be cancelled
oblig   shutdownCancellation {
      on        systemShutdown;
      subject   process/;
      target    printManager;
      action    cancelDoc(Job);
      when      Job.owner == process.owner;
}
```

Their Event Calculus representation shown below can be automatically derived

```
% Authorisation
initiates(requestAction(Process, operation(printMgr,
 cancelDoc(Job))), permit(Process, operation(printMgr,
 cancelDoc(Job))), T) ←
 validSpec(Process, operation(printMgr, cancelDoc(Job)))
 ∧ holdsAt(pos(state(Process, owner, root)), T).

initiates(requestAction(Process, operation(printMgr,
 cancelDoc(Job))), deny(Process, operation(printMgr,
 cancelDoc(Job))), T) ←
 validSpec(Process, operation(printMgr, cancelDoc(Job)))
 ∧ holdsAt(neg(state(Process, owner, root)), T)
 ∧ holdsAt(neg(state(Job, owner, Process)), T).

% Obligation
initiates(systemEvent(systemShutdown), oblig(Process,
 operation(printMgr, cancelDoc(Job))), T) ←
 validSpec(Process, operation(printMgr, cancelDoc(Job)))
 ∧ holdsAt(pos(state(Job, owner, Process)), T).
```

The interaction between these policy rules and the enforcement, and behaviour model is illustrated in Figure 2-3. Here, the initial system state consists of a process, proc1, owned by 'root' and a print job, job1 that is owned by that process. When the systemShutdown event occurs at t=1, this triggers the obligation rule shown above. The assertion of the obligation fulfils the condition of the obligation enforcement rule and causes a request to perform the cancelDoc(Job) action to be generated. This event triggers the evaluation of the first authorisation policy rule above, causing the operation to be permitted, which then satisfies the condition of the access control enforcement rule. After the doAction(…) event occurs at t=5, the termination rules specified in the enforcement model cause the permit and oblig fluents to be terminated at t=6.

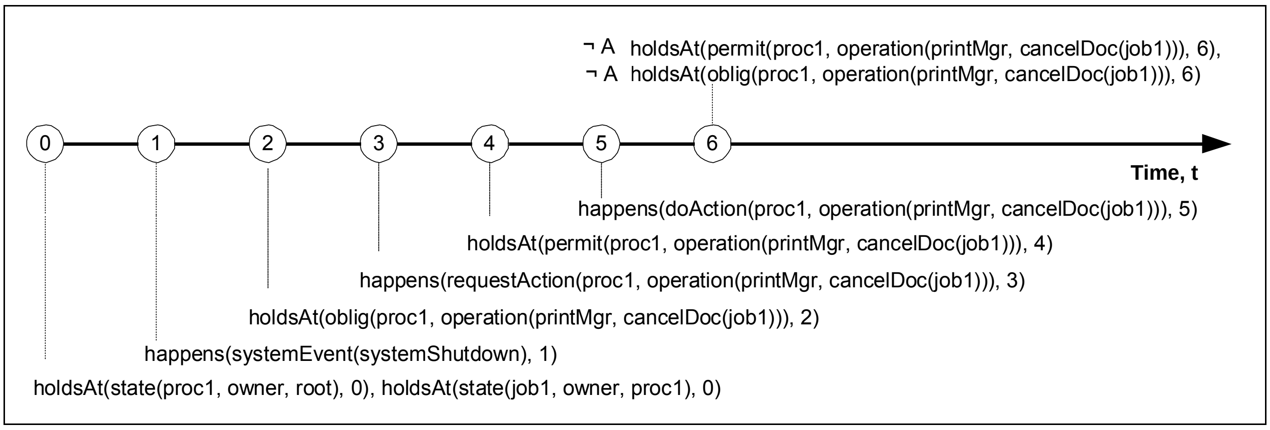

¬ A holdsAt(permit(proc1, operation(printMgr, cancelDoc(job1))), 6),
¬ A holdsAt(oblig(proc1, operation(printMgr, cancelDoc(job1))), 6)
0
1
2
3
4
5
6
Time, t
happens(doAction(proc1, operation(printMgr, cancelDoc(job1))), 5)
holdsAt(permit(proc1, operation(printMgr, cancelDoc(job1))), 4)
happens(requestAction(proc1, operation(printMgr, cancelDoc(job1))), 3)
holdsAt(oblig(proc1, operation(printMgr, cancelDoc(job1))), 2)
happens(systemEvent(systemShutdown), 1)
holdsAt(state(proc1, owner, root), 0), holdsAt(state(job1, owner, proc1), 0)

Figure 2-3 Timeline of Interactions between Policy Rules and Enforcement Model

2.2 Policy Analysis based on abductive reasoning

Since the policy specification notation described above supports policy types that are semantically opposite to each other (e.g. obligations and refrains), conflicting policy specifications could arise. It is therefore important to provide a means of detecting conflicts in the policy specification as part of the logical framework.

The different types of conflicts that can occur in a policy specification are identified in [10]. Modality conflicts arise when two policies are specified using the same subjects, targets and actions but are of opposite modality (e.g. positive and negative authorisations). This type of conflict is domain-independent since conflicts could occur irrespective of the application domain for which the policies are being specified. Other types of conflict identified in the literature fall into the category of application specific conflicts. As described in [11], these include conflicts of duty, conflicts of interest, multiple manager conflicts, conflicts of priorities for resources and self-management conflicts. Considering the types of conflict described above, it is possible to define rules that can be used to recognise conflicting situations in the policy specification.

2.2.1 Modality Conflicts

Modality conflicts involving authorisation policies occur when there are two policies, one an authorisation and the other a prohibition, defined for the same subject, target and action. The `authConflict` predicate defined below holds if an authorisation conflict is detected.

```
holdsAt(authConflict(Subj, Op), Tm) ←
 holdsAt(permit(Subj, Op), Tm) ∧  holdsAt(deny(Subj, Op), Tm).
```

In a similar fashion, rules for detecting conflicts between obligations and refrains; and unauthorised obligation conflicts can be defined as follows:

```
holdsAt(obligConflict(Subj, Op), Tm) ←
  holdsAt(oblig(Subj, Op), Tm) ∧ holdsAt(refrain(Subj, Op), Tm).

holdsAt(unauthObligConflict(Subj, Op), Tm) ←
  holdsAt(oblig(Subj, Op), Tm) ∧ holdsAt(deny(Subj, Op), Tm).
```

In each of these rules, the Op variable will be instantiated with an operation pertaining to the managed objects.

2.2.2 Application Specific Conflicts

One of the most common types of application specific conflict cited in the literature is conflict of duties (alternatively stated as the requirement to ensure separation of duties) [11]. A conflict of duties will arise if the same subject is permitted to perform operations that, in the context of the application, are defined to be conflicting. For example, in a company financial system, the operation of entering a request for payment and the operation of approving that request are potentially conflicting if the same user can perform both operations.

Rules for application specific conflicts must be defined using constraints that include application specific data in addition to policy information. However, before defining rules for detecting such conflicts, it is important to have a means of specifying this application specific information. The description of the various types of application specific conflicts in [11], suggests that:

- A conflict of duty arises when the same subject performs both operations on the same target (e.g. an employee makes a payment request and approves it).
- A conflict of interest arises when the same subject performs each of the operations on different targets. (e.g. a bank provides investment advice to a client whilst performing a merger for a competing client).
- Different subjects perform each of the operations on a single target and the outcome of each operation is incongruent with the other. (e.g. spooling a job to a printer and shutting the same printer down).

In order to capture this application specific information, we extend the system specification language with a new symbol – conflictingOps(ConflictType, [Ops]). Here the ConflictType represents a constant value from the set {conflictDuty, conflictInterest, conflictGoal, conflictSelfMgmt}, indicating the type of application specific conflict that may arise if the operations are used in a policy specification. The members of the Ops list are instantiated using the operation term defined previously. The symbol can be used to define ground literals in the system specification, specifying the action/target object combinations that will potentially conflict. In the case of the conflict of duties example mentioned above, the potential conflict between the operations of requesting a payment and approving a payment would be represented as follows:

```
conflictingOps(conflictDuty, [operation(payment,
request(PaymentID,Amount)),operation(payment,approve(PaymentID))])
```

As described in the literature, the principle of separation of duty can take a number of different forms. In the first case, static separation of duty is ensured by not permitting a subject to perform an operation, Op1, if that subject has ever been granted permission for a different operation, Op2, and Op1 and Op2 are defined as members of a set of conflicting operations. A policy specification that violates this principle will give rise to a conflict of duty. The second variation, dynamic separation of duty, requires that the runtime behaviour of the system should not allow conflicting operations to be performed. Finally, the Chinese Wall policy [12] is a specialised form of dynamic separation of duty that prevents a subject performing any conflicting actions on one target, if the subject has already been given permission to perform a conflicting action on a different target.

In the formalism presented here, we model the dynamic behaviour of the system because this is necessary for dealing with the effects of having constraints in the policy specification. This allows us to treat the detection of static and dynamic conflicts of duty in a similar manner by defining rules of the following form, depending on the number of operations that could cause conflicts:

```
holdsAt(sepOfDutyConflict(Subj, Ops), Tm) ←
 holdsAt(permit(Subj, Op1), T1) ∧
 holdsAt(permit(Subj, Op2), T2) ∧ ... ∧
 holdsAt(permit(Subj, OpN), TN) ∧
 conflictingOps(conflictDuty, Ops) ∧
 memberOf(Op1, Ops) ∧
 memberOf(Op2, Ops) ∧ ... ∧ memberOf(OpN, Ops) ∧
 T1=<T2=<...=<TN=<Tm.
```

The rule for detecting a conflict in a Chinese Wall policy is different because the conflict condition also depends on the targets involved. We represent this as follows:

```
holdsAt(cwConflict(Subj,Target1,Action1,Target2,Action2), Tm) ←
 holdsAt(permit(Subj, operation(Target1, Action1)), T1) ∧
 holdsAt(permit(Subj, operation(Target2, Action2)), T2) ∧
 conflictingOps(conflictDuty, Ops) ∧ Target1 != Target2 ∧
 memberOf(operation(Target1, Action1), Ops) ∧
 memberOf(operation(Target2, Action2), Ops) ∧
 T1 =< T2 =< Tm.
```

Another type of conflict, identified in the literature as a multiple management conflict, arises when different subjects attempt to perform actions on the same target, where the goals of those actions are incongruent. For example, spooling a job to a printer and shutting the same printer down are operations with incompatible goals. We represent these operations using the constant, conflictGoal, in the conflictingOps term. The following is a representation of the printer example above using this symbol:

```
conflictOfGoalsOps(conflictGoal, [operation(printer,
  printDoc), operation(printer, shutDown)]).
```

Once the incompatible operations have been defined, the following rules can be used to identify multiple manager conflicts in a policy specification:

```
holdsAt(conflictOfMultiManagers(Subj1, Subj2, ..., SubjN
                                Ops), Tm) ←
 holdsAt(permit(Subj1, Op1), T1) ∧
 holdsAt(permit(Subj2, Op2), T2) ∧  ...  ∧
 holdsAt(permit(SubjN, OpN), TN) ∧
 conflictingOps(conflictGoal, Ops) ∧
 memberOf(Op1, Ops) ∧
 memberOf(Op2, Ops) ∧  ...
 memberOf(OpN, Ops) ∧ T1 =< T2 =< ... =< TN =< Tm.
```

Similar rules are specified for other types of application specific conflicts, such as conflicts of interest and self-management conflicts.

By using one of the conflict fluents (e.g. unauthObligConflict) as a goal state, it is possible to query the system specification for event sequences that would result in a conflict occurring. If no such sequence can be derived, it can be considered that the policy specification is free of this particular conflict type.

The current implementation of the analysis system makes use of the abductive proof procedure presented in [8]. By treating the conflict fluents as safety properties of the system, this technique reduces the complexity of the abductive proof procedure to two time points – the time before the conflict arises (t) and the time after it arises (t1). Additionally, provided the conflict term is specified using ground literals, it can be shown that the query will always generate a complete explanation for any conflicts and it will always terminate.

The textbox below shows an illustration of performing such a query on the example system presented previously. Here some of the solutions, such as the last, present the trivial case in which a conflict might occur. However, the first solution suggests that there is a sequence of events that will cause a conflict. Therefore, it can be concluded that the policy specification contains a conflict.

```
?- demo([holdsAt(obligConflict(printMgr,
   operation(printer, printDoc)), t1)], [], Plan).
Plan = [initiallyTrue(state(printMgr,state,
  shuttingDown)), happens(systemEvent(printReq),t)]
Plan =  [initiallyTrue(refrain(printMgr,
  operation(printer,printDoc))), happens(systemEvent(printReq),t)]
Plan = [happens(systemEvent(printReq),t),
  initiallyTrue(state(printMgr,state,shuttingDown)),
  initiallyTrue(oblig(printMgr,operation(printer, printDoc)))]
Plan = [initiallyTrue(refrain(printMgr,
   operation(printer,printDoc))),
   initiallyTrue(oblig(printMgr, operation(printer,printDoc)))]
```

2.3 Policy analysis based on Finite State Transducers

It is known that conflict detection and conflict resolution are complex issues, and in the network management context they are especially complex because of the diversity of technologies that must be dealt with. We have been looking for ways to model different kinds of conditions and policies for different technologies in common and, over all, efficient ways. We have chosen to use Finite State Transducers. They have been useful in a wide range of fields, but particularly in Natural Language Processing. This discipline makes intensive use of grammatical rules, which are ambiguous by nature, and require quick decisions based on those rules, in particular in fields as speech recognition, with major performance needs.

Our proposal is an approach consisting of an adapted subset of finite state automata concepts, to detect and solve conflicting policy rules. They are special automata for which, on each edge, there are two labels instead of one.

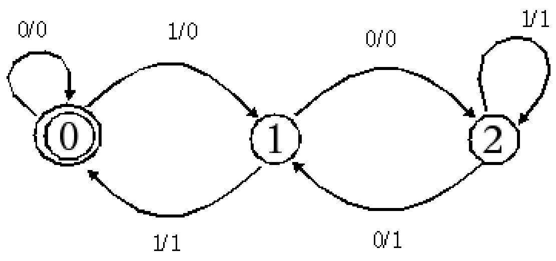


Figure 2-4: Transducer representing division by 3 of binary numbers

A model based on finite state transducers is not a completely new idea, but we think that its full potential has not been exploited. In particular our transducers will not model the network behaviour, but rather the policies themselves.

A finite state transducer is a kind of automaton with two labels on each transition. The first is the classic label for incoming symbols and the second is an output label. With these two labels, the transducer recognises an input string and produces a new output string at the same time.

For representing policies with FSTs we deployed the model presented in [13]. It is based on a modification of predicate augmented FSTs [14], in which predicates were replaced by a metric representing a sort of distance between a policy and a given event. We called this new kind of finite state machines, *Finite State Transducer with Tautness Functions and Identities* (TFFST).

<u>Obligations</u>

Starting to exemplify how our model works, an obligation as simple as *“if the user dials up then execute the action connect”* is modelled as shown in Figure 2-5. The condition *“user dials up”* is this “d” here, and the action *“connect”* is this “k” here.

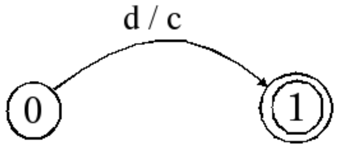


Figure 2-5: An obligation expressed as a transducer

They are modelled as a graph with one edge only where the incoming label is the condition part and the outgoing label is the action. It must be added to the existing model with the *union* operation described later.

TFFST Operations

The most interesting aspect of classic FSTs is the operations between them. We have concentrated our work on developing the algorithms for performing these operations on TFFSTs in the context of policy-based management. This restriction is important because, in the general case, some of the most useful operations do not result in valid transducers.

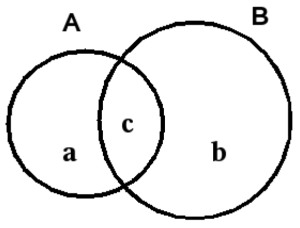
Classic operations on FSTs include: **union**, **intersection**, **complement**, **composition**, **Kleene closure** and **determinisation.**

Determinisation

Among those operations, determinisation is the one that carries out the conflict resolution process. It consists of transforming a non deterministic transducer in a deterministic one. In Figure 2-7, if we look at node zero, if the incoming event fulfils condition “a,” we do not know which path must be chosen and both must be probed. This is the first problem and is an efficiency problem. In the case that conditions “e” and “g” overlap, we have another problem, a conflict situation. Both paths are valid, and this is what we are trying to avoid. Therefore, to choose between different options, instead of simple conditions on the transitions, we have placed on them something that express a notion of distance or condition tautness around an event. We called that *tautness function*, in this way we have a little more information for making decisions.

Tautness Functions

We have defined a metric that, besides saying when an event fulfils a condition, gives us some information about how far from the event or, in other words, how taut the condition is. The value of these functions must be in the range [-1,1]. Based on the algebra for fuzzy sets, we have defined an algebra for these tautness functions allowing us to compare any two conditions. Tautness Functions (TF) are an abstraction layer making the conflict resolution process as general as possible. From the point of view of the algorithms, they are technology-independent and make it possible to deal with orthogonal conditions. In Figure 2-6 you can see some basic rules that TFs follow. Still, they are a major research issue in themselves and the viability of the model depends on their viability.



$$\tau_A(a) \geq 0$$
$$\tau_A(b) < 0$$
$$\tau_A(c) < \tau_B(c)$$
$$\tau_{A \vee B} = max(\tau_A, \tau_B)$$
$$\tau_{A \wedge B} = min(\tau_A, \tau_B)$$
$$\tau_{\neg A} = -\tau_A$$

$$\tau_{A \rightarrow_\tau B} = \begin{cases} \tau_A, & if\ \tau_A < \tau_B \\ -1, & else \end{cases}$$

$$\tau_{A \rightleftarrows_\tau B} = \begin{cases} \tau_A, & if\ \tau_A = \tau_B \\ -1, & else \end{cases}$$

Figure 2-6: Tautness Functions

For example when computing a condition comparing the bandwidth of two networks such as the one below:

```
      …      when t.effectiveBW([nic A]) < t.effectiveBW([nic B]); …
```

If `nic A` is connected to a hotspot with a maximum data rate of 11 Mb/s and `nic B` uses a GSM/GPRS network with a maximum data rate of 144 kb/s, and assuming their values have uniform distributions, when we evaluate the condition to true, its tautness function is:

$$\frac{144kb/s}{111Mb/s \times 1000} \times 0.5 = 0.00654$$

A value as close to zero as this one means a very strong condition. Hence, it is very unlikely that this situation will occur and the manager must have had a very good reason to specify a policy with this condition.

The algorithm

The actual conflict resolution process is carried out by the determinisation algorithm, in Figure 2-7 we see a FST before transformed. If an incoming event fulfils condition “a,” there are two possible paths to take - which means two possible actions to trigger. But notice that if the next event fulfils condition “e” but not condition “g,” the supposed ambiguity does not exist. Thus, the first step is to delay the decision for as long as possible.

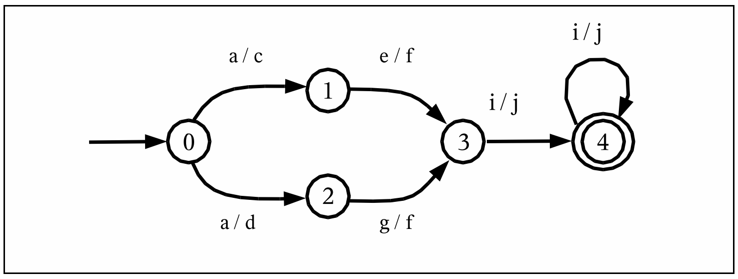


Figure 2-7: Before determinisation

If you see Figure 2-8, there is an epsilon label here indicating that we will not produce any action for now. Then, a transition is created for each possible combination of conditions, for example, the upper edge will be followed when the incoming event fulfil “e” but not “g,” so we have enough information to throw an event “c” and an event “f”.

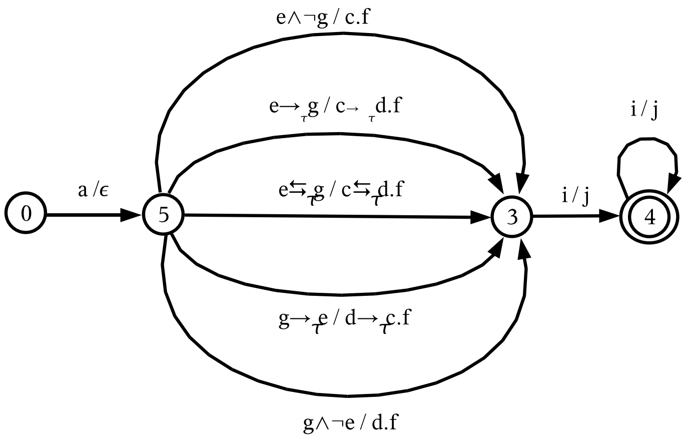


Figure 2-8: After determinisation

Other transitions are for example for events to which "e" is more taut than "g," or for situations in which we cannot say if one condition is more taut that another on the event. In this case, if the two actions are compatible, we are triggering both; if not, we dispatch an error. However, this transition may be noted for the operator in compilation time, and some action may be taken to resolve the situation.

The conflicts we are working on are: those that can be detected and resolved in compilation time (known mostly as modality conflicts); and those for which we can foresee the occurrence of a dynamic conflict. This algorithm does not solve unsolvable problems, but a framework is given for working efficiently and independently of technology. Moreover, conflict resolution is carried out beforehand and the model takes advantage of experience from other research fields like Natural Language Processing.

2.3.1 Application to multi-domain pricing scenarios

In a multi-provider mobile operator network we observe different prices for a given service. These prices are determined by the operators and currently there is no governing body for price regulation; in brief the operators are free to select any price for a given service as they prefer. In order to determine the price of a service, the service provider must select a pricing schema and likely he must set representative parameters of this pricing schema. For simplicity, let's assume only two providers of mobile Internet access as shown in Figure 2-10. Also for simplicity let's assume that both service providers use the same pricing model that is stated as a linear combination of three variables x, y, and z. These variables are weighted by three parameters a, b and c respectively. As said before, each provider has its own policy or policies to set the values of the parameters a, b and c whereas x, y and z are considered as context variables that are properly monitored. We also assume that in the area of common coverage the conditions by means of which a provider determines the parameters in the formula are the same. Therefore a conflict may occur when these two service providers use different rules to set these parameters.

As a consequence, when there is a conflict, the price of a given service for the same user is different if it is offered by one or the other service provider. We attempt to solve this conflict in order to determine a unique price, an optimum or fare price that would be offered to both service providers for their own reference. The entity entrusted to solve the conflict and therefore evaluate the fair price is the Pricing Manager also reflected in Figure 2-10.

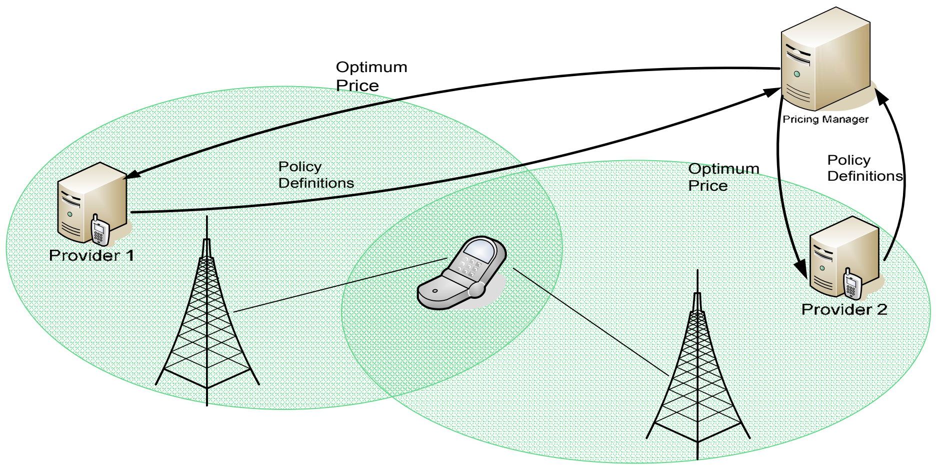


Figure 2-9: Multidomain Internet access scenario

Pricing formula

As said before, each provider is using the same pricing model to determine the price P of a given service as follows

$P = a(x) + b(y) + c(z)$

Where a, b and c are the price components which in turn depend of the context variables x, y and z respectively. We will describe our formula using the following picture.

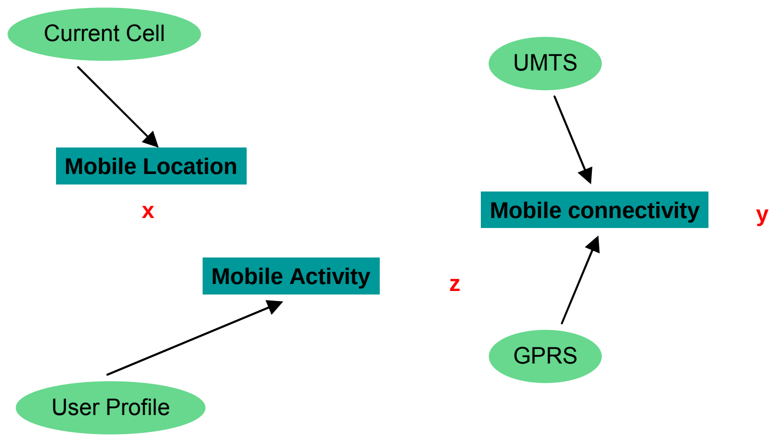


Figure 2-10: Intervening context variables

Interpretation and values for variables

In the formula of P we have to determine the values of the corresponding variables. Table 2-3 shows the values adopted in this example. These values are binary but can be multivalent as well.

Mobile location (variable X)	Value
Current cell is busy	
Current cell is not busy	
We assume there is a monitoring mechanism allowing to determine if a given cell (or coverage area) is busy or not	
Mobile connectivity (variable Y)	Value
The terminal makes use of a GPRS connection (also called type-1)	
The terminal makes use of a UMTS connection (also called type-2)	
Mobile activity (variable Z)	Value
The user is qualified as a “High-end” user	
The user is qualified as a “Low-end” user	

Table 2-3: Values to be adopted by the context variables

2.3.1.1 Pricing policies

Pricing policies determine the values of parameters a, b and c according to the values of the context variables x, y and z as follows

Policies for Provider 1

If the value of x is busy, value of a is 0.8, and if it is not busy the value of a is 0.3

If y is of type1 then b is 0.5 and if y is of type2 then b is 0.4

If z is of type high end user, then c is 0.1, otherwise its 0.4.

These values represent direct costs which will be computed to give the price for a service, by provider 1. Now let us see the policies that provider1 implies on the network. Accordingly he writes down certain pricing policies based on this information

Policy1: Alice is in busycell, calling on a GPRS network and is a high-end user.
Corresponding price for this would be 0.8 + 0.5 + 0.1.

Policy2: Alice is in busycell, calling on a GPRS network and is a low-end user.
In this case the price would change 0.8 + 0.5 + 0.4.

Policy3: Alice is in busycell, calling on a UMTS network and is a high-end user.
The price in this case will be 0.8 + 0.4 + 0.1.

Policy4: Alice is in busycell, calling on a UMTS network and is a low-end user.
Price in this case =0.8 + 0.4 + 0.4.

Policy5: Alice is not in busycell, calling on a GPRS network and is a high-end user.
Price in this case =0.3 + 0.5 + 0.1.

Policy6: Alice is not in busycell, calling on a GPRS network and is a low-end user.
Price in this case =0.3 + 0.5 + 0.4.

Policy7: Alice is not in busycell, calling on a UMTS network and is a high-end user.
Price in this case =0.3 + 0.4 + 0.1.

Policy8: Alice is not in busycell, calling on a UMTS network and is a low-end user.
Price in this case =0.3 + 0.4 + 0.4.

Policies for Provider 2

If the value of x is true, value of a is 0.5, and if it is false the value of a is 0.3

If y is of type1 then b is 0.6 and if y is of type2 then b is 0.6

If z is of type high end user, then c is 0.4, otherwise its 0.4

Policy1: Alice is in busycell, calling on a GPRS network and is a high-end user.
Corresponding price for this would be 0.5 + 0.6 + 0.4.

Policy2: Alice is in busycell, calling on a GPRS network and is a low-end user.
In this case the price would change 0.5 + 0.6 + 0.4.

Policy3: Alice is in busycell, calling on a UMTS network and is a high-end user.
The price in this case will be 0.3 + 0.6 + 0.4.

Policy4: Alice is in busycell, calling on a UMTS network and is a low-end user.
Price in this case would be 0.3 + 0.6 + 0.4.

Policy5: Alice is not in busycell, calling on a GPRS network and is a high-end user.
Price in this case would be 0.3 + 0.6 + 0.4.

Policy6: Alice is not in busycell, calling on a GPRS network and is a low-end user.
Price in this case would be 0.3 + 0.6 + 0.4.

Policy7: Alice is not in busycell, calling on a UMTS network and is a high-end user.
Price in this case would be 0.3 + 0.6 + 0.4.

Policy8: Alice is not in busycell, calling on a UMTS network and is a low-end user. Price in this case would be 0.3 + 0.6 + 0.4.

Summary of policies applicable in the scenario

Policy	BusyCell Value	Connection Type Value	User Profile Value	Total cost	Possible conflict
Policy1	True	GPRS	High-end	0.8 + 0.5 + 0.1	a b c
Policy2	True	GPRS	Low-end	0.8 + 0.5 + 0.4	a b
Policy3	True	UMTS	High-end	0.8 + 0.4 + 0.1	a b c
Policy4	True	UMTS	Low-end	0.8 + 0.4 + 0.4	a b
Policy5	False	GPRS	High-end	0.3 + 0.5 + 0.1	b c
Policy6	False	GPRS	Low-end	0.3 + 0.5 + 0.4	b
Policy7	False	UMTS	High-end	0.3 + 0.4 + 0.1	b c
Policy8	False	UMTS	Low-end	0.3 + 0.4 + 0.4	b

Table 2-4: Policy table for provider1

Policy	BusyCell Value	Connection Type Value	User Profile Value	Total cost	Possible conflict
Policy1	True	GPRS	High-end	0.5 + 0.6 + 0.4	a b c
Policy2	True	GPRS	Low-end	0.5 + 0.6 + 0.4	a b
Policy3	True	UMTS	High-end	0.5 + 0.6 + 0.4	a b c
Policy4	True	UMTS	Low-end	0.5 + 0.6 + 0.4	a b
Policy5	False	GPRS	High-end	0.3 + 0.6 + 0.4	b c
Policy6	False	GPRS	Low-end	0.3 + 0.6 + 0.4	b
Policy7	False	UMTS	High-end	0.3 + 0.6 + 0.4	b c
Policy8	False	UMTS	Low-end	0.3 + 0.6 + 0.4	b

Table2-5: Policy table for porvider2

In the *Possible Conflict* column we have the parameters a, b or c that may be in conflict. For instance, in row 1 of Table 2-5, we have in the column the three parameters a, b and c, meaning that the weighting factors a, b and c are not the same for the two providers, which is a conflict that our Pricing Manager has to solve according to the conflict resolution mechanism above described.

2.4 Conclusions

In this chapter we have described the use of Event Calculus and abductive reasoning for formalising both authorisation and management policies and performing analysis of policy-based systems. In addition, an alternative method involving Finite State Transducers has also been presented for policy analysis. Although our focus has been primarily on conflict detection, checking that a given policy specification satisfies well-defined properties follows an identical pattern to the check for application specific conflicts. The formal representation proposed is sufficiently expressive to model systems using a combination of authorisation, obligation and refrain policies. We have shown how an abductive analysis procedure has been used to analyse the policy specifications and how this procedure identifies the specific circumstances under which conflicts occur in addition to identifying the conflicts themselves. Although the underlying formal representation and reasoning are not easily accessible to users that are not well versed in logic programming, the tools we have implemented hide the underlying complexity and automatically derive the formal representation from the policies themselves and from design-level models of managed objects' behaviour such as state charts. This analysis technique has been implemented and integrated with a refinement approach thus providing an integrated tool set for analysis and refinement. Further information and a demonstration of the implementation are available at: http://ponder2.net/cgi-bin/moin.cgi/PonderARTProject

3 XML and Web Services management technologies

3.1 Introduction

One of the most important technologies to consider for the realisation of Autonomic Management solutions are Web Services (WS) and XML related approaches. Although these technologies have been used for years, recent technological advances and optimisation efforts have made them very attractive for managing complex networks. Considering the fact that the management of networks and systems still relies on protocols and technologies that were developed several decades ago, like CLI and SNMP, the new emerging paradigm of Web Services-based Management is gaining pace. In this sense, Web Services can be an appealing enabler of Autonomic Management.

The concepts of Autonomic Management [1,2] can benefit from Web Services-based approach by exploiting their unique features. Self-configuration can be significantly simplified if build on a platform independent and interoperable framework based on Web Services. By exploiting XML, a WS-based solution takes the burden of communicating with the diversity of different combinations of Operating Systems and CPU architectures. This diversity is a common feature of heterogeneous next generation networks, a target area for the deployment of Autonomic Management. In addition, self-awareness and self-optimisations can be catered by a WS-based monitoring and notifications framework, assisted by XML-based ontologies schemes for context collection.

Technologies based on XML, including Web Services, can offer a flexible environment to develop and deploy autonomic systems. They can provide a rich communications framework that goes over the limitations of traditional management protocols and is open to a large developer community. Tools and development environments used for exploiting XML's potentials have significantly matured and their performance has been improved vastly recently. Research has shown that management based on Web Services can match and in many cases exceed the performance of traditional protocols like SNMP. Results are presented in following sections, measuring and comparing bandwidth use, latency, CPU utilisation and memory usage under various conditions.

In addition, these technologies receive intense standardisation efforts and try to establish a transparent and interoperable framework. Today they can be found even on lightweight devices (e.g. PDAs and mobile phones) with limited capabilities. The platform independence of XML technologies is a major advantage for the increasingly heterogeneous networks. This is an issue of paramount importance when considering autonomic solutions, especially for wireless ad hoc networks. As explained in subsequent sections, Web Services offer a variable level of transparency and generality, allowing the system designer to highly customise the communication interfaces to match diverse requirements.

An illustrative example of a Web Service-based network monitoring system is presented to demonstrate the use of Web Services for integrated management. The example is based on the SNMP Interface table (`ifTable`), which is part of the Interfaces Group MIB (`IF-MIB`) [15] and records the comparative performance results

3.2 Overview of Network Management [1]

It is interesting to observe that the management of networks and systems still relies on protocols and technologies that were developed several decades ago. The most important management tool, for example, still seems to be the Command Line Interface (CLI), which was introduced in the early days of networking to manage the very first systems. Currently operators still rely on this interface to configure their systems; the CLI is not only used to manually login to remote systems to monitor and modify the operation of these systems, but also used by fully automated management processes that are implemented in the form of scripts. In fact such scripts are essential to run networks and systems in an effective and cost efficient way; automated scripts are vital to keep operational costs low.

Unfortunately the syntax of the CLI has not been standardized; it varies from vendor to vendor and sometimes within a vendor between different product lines or even different versions. Scripts are therefore hardly portable, and considerable investments are needed to maintain these scripts. It is therefore not surprising that already in the eighties of the previous century organizations like the International Organization for Standardization (ISO) or the Internet Engineering Task Force (IETF) made proposals for standardized management protocols. The most successful of these protocols is the Simple Network Management Protocol (SNMP), which is accompanied by a standard describing the Structure of Management Information (SMI) and a standard defining the objects within the Management Information Base (MIB-II). Since this MIB-II defined only a small number of objects, soon proposals for additional MIB modules appeared. The first version of these SNMP standards were developed in a very short time frame and became full standard in 1990; very soon they got implemented in many products and tools and used on a wider scale than originally expected. Because of this success, the original plans to migrate to ISO management protocols were abandoned and a decision was made to create a second version of the SNMP standards; such version should have better information modeling capabilities, improved performance and, most importantly, stronger security. The work on information modelling progressed relatively smoothly and resulted in 1999 in a new SMIv2 standard. Also there was soon agreement on the introduction of a new PDU to retrieve bulk data. Unfortunately, the original developers of SNMPv2 disagreed on a security model and failed to find a compromise. SNMPv2 was therefore standardized without strong security and a new group, with different people, had to be created to solve SNMP security. After many years of delay this resulted in SNMPv3, which eventually became full standard in 2003. At that time a new IETF group was formed, called the Evolution of SNMP (EoS) Working group, but also this group could not agree on any improvements of SNMP. Evolution of SNMP therefore failed [16]. Although SNMP is widely used for monitoring, managers still have to rely on the CLI and hand made scripts. To solve this problem, revolutionary approaches to management are needed.

The first questions to ask before proposing a new approach to management is: *do we need dedicated protocols to exchange management information, or can we use existing protocols for this purpose?* To answer this question, one has to investigate which

1 This Section contains a summary of the chapter *"What Can Web Services Bring To Integrated Management?"*. The original chapter has as authors: Aiko Pras and Jean-Philippe Martin-Flatin, and is included in the *"Handbook of Network and System Administration"*, editors: Jan Bergstra and Mark Burgess, ISBN: 13 978-0-444-52198-9, Copyright: Elsevier (2007). Copied with permission from Elsevier.

functionality is needed to exchange management information. In general this functionality involves reading and modifying (management) information on remote systems; this is quite similar to what is needed, for example, for finding flight information and booking hotels. In fact there are no good reasons to invent new management protocols; current middleware technology is perfectly able to do this job.

Although several middleware technologies have been proposed in the previous decade (like Corba and JAVA), the most important technology seems to be Web Services. This technology can be used in combination with virtually all programming languages, is supported by many development platforms (e.g. .Net, Sun-One, JBuilder), is included in all major operating systems and easily integrates with applications like Microsoft Office. In fact, calling Web Services from spreadsheets like excel is very easy, and the idea of building network and service management scripts within spreadsheets is challenging. An additional advantage of using Web Services for management is that there are many skilled developers in this area and many tools. The implementation of Web Services based management applications is therefore much easier than developing SNMP based applications.

3.3 Web Services Management Approaches

At first sight, an intuitive approach to introduce Web Services for network management would be to map existing management operations (e.g., the SNMP operations `get`, `set` and `inform`) onto Web Services. Is this efficient? Is this the way Web Services should be used?

Mapping the current habits in SNMP-based network management onto Web Services leads to four possible approaches, depending on what we call the genericity and the transparency chosen [17]. Genericity characterizes whether the Web Service is generic (e.g., a Web Service that performs a `get` operation for an OID passed as parameter) or specific to an OID (e.g., `getIfInOctets`). Transparency is related to the parameters of the Web Service, which can be either defined at the WSDL level or kept as an opaque string that is defined as part of a higher-level (XML) schema. This will be referred to as Web Services that have either transparent or non-transparent parameters. The four resulting forms of Web Service are depicted in Figure 3-1 .

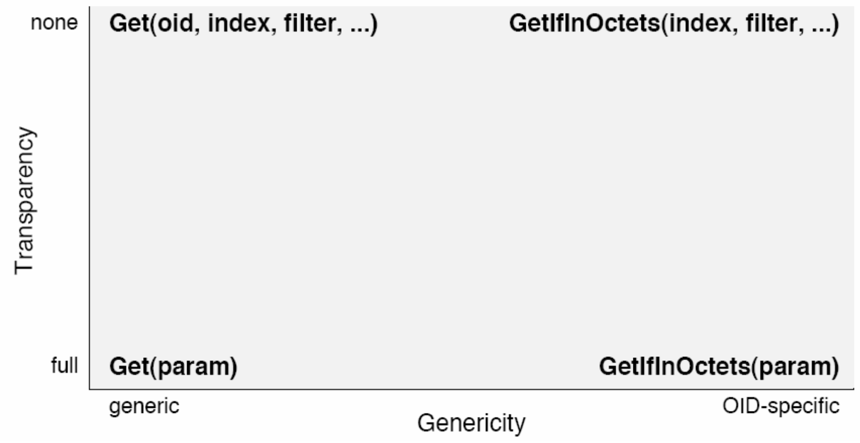
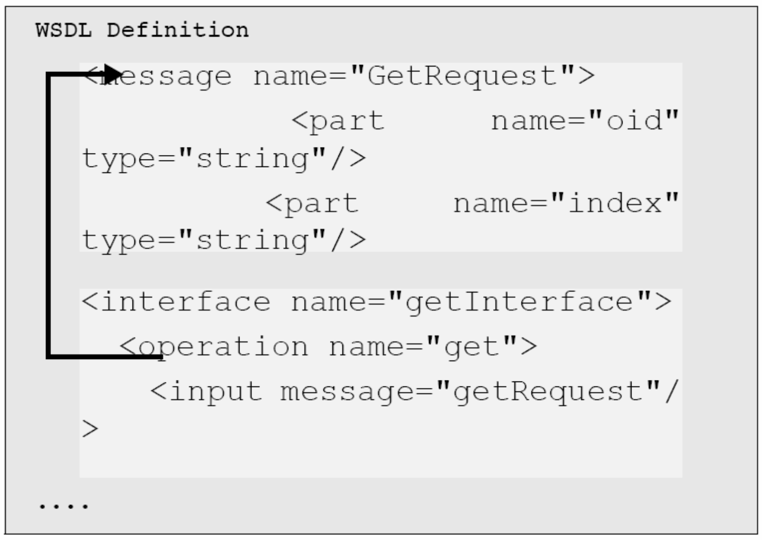


Figure 3-1 : Fine-grained Web Services: transparency vs. genericity

Parameter Transparency

Web Service operations are defined as part of the abstract interface of WSDL definitions. An operation may define the exchange of a single message, such as a `trap`, or multiple messages, such as `getRequest` and `getResponse`. For each message the parameters are defined in so-called <part> elements. One approach would be to define multiple parameters, and define for each parameter its syntax in a separate <part> element. An example of this is a `GetRequest` message, which defines three parameters: oid (object identifier), index and filter. This example of non-transparent parameters is shown in Figure 3-2 ; note that for simplicity in this example all parts are of type string.

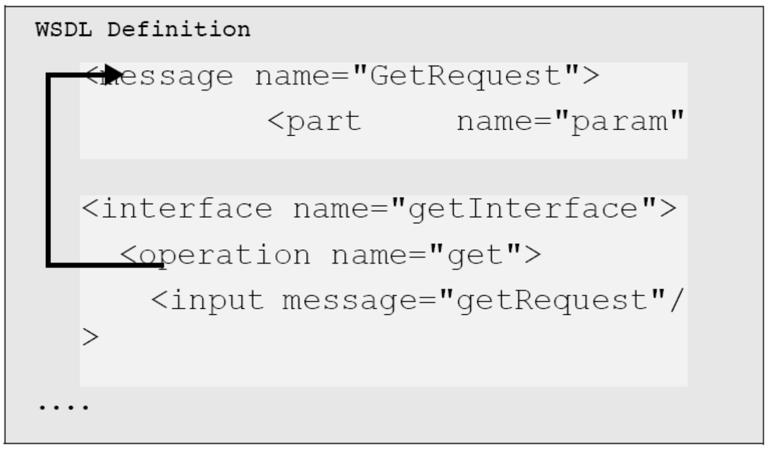


```
WSDL Definition
<message name="GetRequest">
           <part      name="oid"
type="string"/>
          <part     name="index"
type="string"/>

<interface name="getInterface">
  <operation name="get">
    <input message="getRequest"/
>
....
```

Figure 3-2: Operation with non-transparent parameters

Another approach would be to define just a single parameter, and leave the interpretation of this parameter to the (manager-agent) application, which resides on top of WSDL layer. In this case only a single <part> element is needed. The syntax of the parameter would be of type string, which means that the parameter is transparently conveyed by the WSDL layer and no checking is performed at that layer. Figure 3-3 provides an example.



```
WSDL Definition
<message name="GetRequest">
          <part     name="param"

<interface name="getInterface">
  <operation name="get">
    <input message="getRequest"/
>
....
```

Figure 3-3: Operation with transparent parameters

An advantage of having parameter transparency, is that a clear separation is made between the management information, and the protocol that exchanges such information. In this case it is possible to change the structure of the management information, without touching the definition of management operations. From a standardization perspective, this may be an attractive feature. Note however that this is not a mandatory feature, since WSDL supports other constructs, like the import statement, to divide a specification over several documents, which can than be progressed independently on the standardization track.

In case of transparent parameters, the application on top of WSDL might use XML encoding and exchange its data as an XML document. In this case an XML parser is needed to extract the parameters from the document. When the application receives such a document, XPath expressions may be used to select the required information. On the manager side, special functionality is needed to create such XML document. Since specific software is needed on top of the WSDL layer, transparent parameters should only be used in case of either more specific network management applications or by more experienced users/developers. In other words, parameter transparency allows the use of powerful and flexible XML software packages on top of WSDL, but may only be interesting for professional users who need this kind of flexibility. For a PC user in his home environment who wants to include some management information in an Excel spreadsheet, this would become too complicated. In such case a simple approach is required, in which the user does not need to parse XML documents with tools like XPath and XQuery. Instead, parsing and checking should be performed at the WSDL layer, thus by the spreadsheet itself. Non- transparent parameters are therefore better suited for simple management applications, such as needed in home environments.

Genericity

Another approach is to vary the genericity of operations from very coarse to very fine. This approach can be illustrated by means of an example. Assume a managed system provides system information, such as `SysLocation` and `SysUptime`, as well as network interface information, such as `IfInOctets`, `IfOutOctets`, `IfInErrors` and `IfOutErrors`. Since the system can have multiple interfaces, the last kind of information can be provided multiple times. Figure 3-4 shows this information in the form of a containment diagram.

To retrieve this information, management operations need to be defined. If these operations are defined at a fine granularity level, a dedicated operation is required for each variable. The resulting operations will than look like `getSysUptime` or `getIfOutOctets` (see Figure 3-4). Note that the operations the retrieve network interface information, need to have a parameter supplied to identify the interface.

Management operations can also be defined at a coarser level of granularity. For example, a `getSystem` operation can be defined to retrieve `SysLocation` as well as `SysUptime`, and a `getInterface(index)` operation to retrieve all management information for a specific interface. The extreme form would be a single `getAll` operation to retrieve all information contained in the managed system.

An important feature of this approach is that the naming of the operations precisely defines the functionality. Therefore it is very clear to the manager to determine which operation to call to retrieve certain information. Note that the number of parameters associated with each operation may be small, since an object identifier is not needed because the name of the operation already identifies the object. Only in case multiple instances of the same object class exist, such as the `Interface` object in the example of Figure 3-4, the manager has to provide an instance identifier.

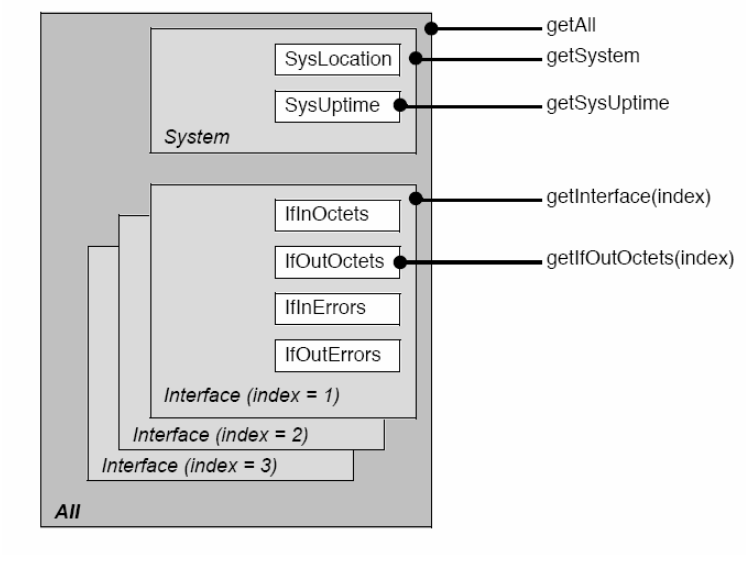


Figure 3-4: Containment diagram

Like in other management approaches (OSI, TMN), the containment hierarchy can be presented as a tree (Figure 3-5). Fine grained operations can be used to retrieve the leaves of the tree; course grained operations to retrieve the leaves as well as their higher level nodes. Note that this is different from SNMP, where only the leaves of the MIB trees are accessible. In general it is not possible, however, to select in a single operation objects from different trees; if the manager wants to retrieve, for example, both `System` as well as `IfInErrors`, two separate operations are needed.

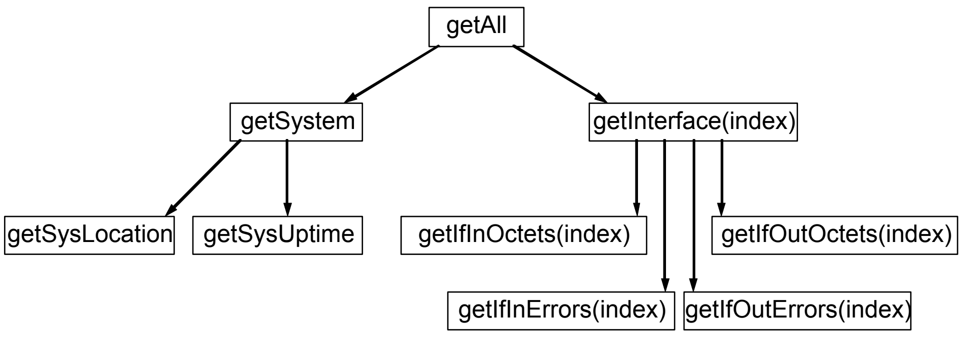


Figure 3-5: Containment tree

To restrict an operation to specific objects, it is possible however to use web services in combination with filtering. Theoretically filtering allows usage of `getAll` to retrieve every possible variable, provided an appropriate filter (and possible index) is given. In combination with filtering `getAll` can thus be considered as an extreme coarse operation, useful for all kind of tasks. In practice, however, filtering can be expensive. If it is performed at the agent side, it may consume too many CPU cycles and memory. If filtering is performed at the manager side, bandwidth may be wasted and the manager station may become overloaded. Although filtering is very flexible and powerful, it should be used with care!

3.3.1 Example of Web Service-Based Management

To illustrate the use of Web Services for integrated management, this section presents an example of a Web Service-based network monitoring system. The example is based on the SNMP Interface table (`ifTable`), which is part of the Interfaces Group MIB (`IF-MIB`) [6]. The `ifTable` provides information on the operation and use of all network interfaces available in the managed system. Such interfaces can be physical interfaces, like Ethernet and WLAN cards, as well as virtual interfaces, like tunnels and the loop-back interface. For each interface the table includes a separate row; the number of rows therefore equals the number of interfaces. The table consists of 22 columns to present information like the interface index (`ifIndex`), the interface description (`ifDescr`), the interface type (`ifType`) etc. Figure 3-6 shows a summary of this table.

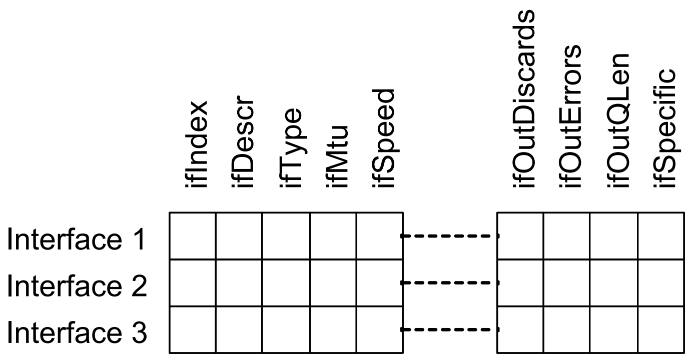


Figure 3-6: Interface Table

Before a WSDL specification for the Interface table can be developed a choice must be made with respect to the genericity of the WSDL operations. In fact three levels of genericity are possible, ranging from very coarse to very fine.

At the highest level of genericity a single course grained operation can be defined to retrieve all objects of the `ifTable` within a single call. This Web Services operation will be called `getIfTable`.

Instead of retrieving the entire `ifTable` at once, it is also possible to define Web Services operations that retrieve only a single row or a single column. The operation that retrieves a single row will be called `getIfRow`. This operation retrieves all information related to a single network interface and requires as parameter the identifier of that interface; a possible choice for that parameter would be the value of `ifIndex`. The operation that retrieves a single column will be called `getIfColumn`. To identify which column should be retrieved, a parameter is needed that can take values like `ifIndex`, `ifDescr`, `ifType` etc. Note that alternatively it would be possible to define separate operations per column; in that case the resulting operations would look like: `getIfIndex`, `getIfDescr`, `getIfType` etc.

Finally, at the lowest level of genericity, separate fine grained operations can be defined to retrieve each individual `ifTable` object. This approach is somehow comparable to the approach used by the SNMP `get` operation (although the SNMP `get` operation allows multiple objects to be retrieved within a single message). Depending on the required parameter transparency, again two choices are possible. The first possibility is to have a generic `getIfCell` operation; this operation requires as input parameters an interface identifier to select the row (interface), as well as a parameter to select the column (like `ifIndex`, `ifDescr`, `ifType` etc.). The second possibility is to have special operations for each column; these operations require as input parameter the identifier of the required interface. An example of such operation would be `getIfType(interface1)`.

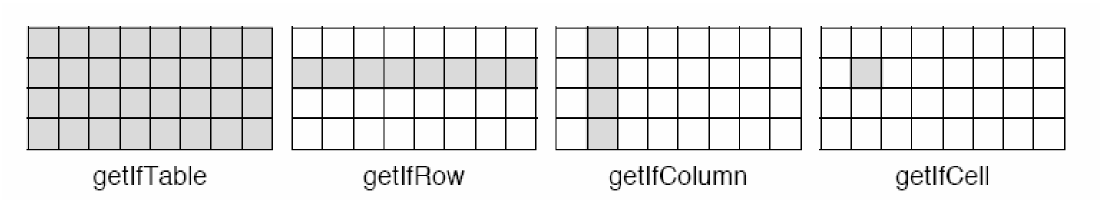


Figure 3-7: Different choices to retrieve table objects

Figure 3-7 summarizes the choices that can be made to retrieve table information; the choices range from very coarse (`getIfTable`) to very fine (`getIfCell`). Note that the figure does not show the choices that are possible with respect to parameter transparency.

As example, the remainder of this section will discuss some main parts of the WSDL definitions of these operations. These definitions, like all WSDL version 2.0 definitions, consists of the following (container) elements:

- The <types> element, to define new data types.
- The <message> elements, to specify the data that belongs to each massage.
- The <interface> element, to combine one or more messages and ease the generation of programming stubs (the previous WSDL version 1.1 used the term portType for this element).
- The <binding> element, to associate the interface element with a transport protocol.
- The <service> element, to associate the interface element with a web address (URL).

The <types> element

XML allows two kinds of type elements:

- `<simpleType>`: An element that contains a single value of a predefined form; an example is an integer or a string.
- `<complexType>`: An element that groups other elements; an example is an address element, which contains street, postal code, and city (sub)elements.

The WSDL definitions of our examples use `<complexType>` elements to group certain object within the `ifTable`. Which object are taken together depends on the genericity of the Web service. For example, all columns of the `ifTable` can be grouped together, yielding a single `<complexType>` that can be used to retrieve entire `ifTable` rows. This complex type will be called `ifEntry` (see Figure 3-8), and can be used by the `GetIfTable` as well as `GetIfRow` operations. Note that for the `GetIfRow` operation, the `<sequence>` element should be left out, since only one row will be retrieved at a time.

```
<types>
  <complexType name="ifEntry">
    <sequence>
      <element name="ifIndex" type="xsd:unsignedInt"/>
      <element name="ifDescr" type="xsd:string"/>
      <element name="ifType" type="xsd:unsignedInt"/>
      <element name="ifMtu" type="xsd:unsignedInt"/>
      <element name="ifSpeed" type="xsd:unsignedInt"/>
      <element name="ifPhysAddress" type="xsd:string"/>
      <element name="ifAdminStatus" type="xsd:unsignedInt"/>
      <element name="ifOperStatus" type="xsd:unsignedInt"/>
      <element name="ifInOctets" type="xsd:unsignedInt"/>
      <element name="ifInUcastPkts" type="xsd:unsignedInt"/>
      <element name="ifInDiscards" type="xsd:unsignedInt"/>
      <element name="ifInErrors" type="xsd:unsignedInt"/>
      <element name="ifOutOctets" type="xsd:unsignedInt"/>
```

Figure 3-8: ifEntry type

The <message> elements

The `<message>` elements are used to the describe the information that is being exchanged between a Web Service and its users. There are `<message>` elements for both request (input) as well as response (output) messages. Such messages consists of zero or more `<part>` elements. In requests message the `<part>` elements represent the parameters of the Web Service. For response messages, the `<part>` elements describe the response data.

The `getIfTable` Web Service supports a single operation only: retrieving the complete table. Figure 3-9 shows the `<message>` elements for this operation. The figure shows a request message containing a “community” string, which is used for authentication purposes, similar to the SNMPv1. The response message contains an element of type `ifEntry`, which was defined in Figure 3-8 , as well as an integer, representing the number of table rows.

```
<message name="GetIfTableRequest">
  <part name="community" type="xsd:string"/>
</message>

<message name="GetIfTableResponse">
```

Figure 3-9: Message definitions for getIfTable

All other Web Services in this section will support multiple operations. The `getIfRow` Web Service can be defined to support two operations, i.e., retrieving a specified row and retrieving a list of valid row index numbers (corresponding to the network interfaces in the agent system). The description of `getIfColumn` and `getIfCell` Web Services can be defined in terms of many operations: for each `ifTable` column a separate operation.

Figure 3-10 shows the messages for two of the operations used by the `getIfCell` Web Service. The request messages have an index element for referencing the correct cell.

```
<message name="getIfIndexRequest">
  <part name="index" type="xsd:unsignedInt"/>
  <part name="community" type="xsd:string"/>
</message>

<message name="getIfIndexResponse">
  <part name="ifIndex" type="xsd:unsignedInt"/>
</message>

<message name="getIfDescrRequest">
  <part name="index" type="xsd:unsignedInt"/>
```

Figure 3-10: Message definitions for getIfCell

The <interface> element

A `<interface>` element defines the operations that are supported by the Web Service. Each operation consists of one or more messages; the `<interface>` element therefore groups the previously defined `<message>` elements into `<operation>` elements. In addition, the `<interface>` element may define `<description>` elements. In general, four types of operations exist: one way, request-response, solicit response and notification. The examples presented in this section are all of the request-response type. Figure 3-11 shows the `<interface>` element for the `getIfTable` Web Service.

```
<interface name="GetIfTableServicePortType">
  <operation name="GetIfTable">
    <documentation>function utMon__GetIfTable</documentation>
    <input message="tns:GetIfTableRequest"/>
    <output message="tns:GetIfTableResponse"/>
```

Figure 3-11: Interface definition for getIfTable

The other Web Services support multiple operations; Figure 3-12 shows part of the `<interface>` element for the `getIfColumn` Web Service.

```
<interface name="GetIfColumnServiceInterface">
  <operation name="getIfIndex">
    <documentation>function utMon__getIfIndex</documentation>
    <input message="tns:getIfIndexRequest"/>
    <output message="tns:uIntArray"/>
  </operation>
  <operation name="getIfDescr">
    <documentation>function utMon__getIfDescr</documentation>
    <input message="tns:getIfDescrRequest"/>
    <output message="tns:stringArray"/>
  </operation>
  <operation name="getIfType">
```

Figure 3-12: Interface definition for getIfColumn

The <binding> element

A `<binding>` element specifies which protocol is being used to transport the Web Service information and how this information is being encoded. Similar to the `<interface>` element, it also includes the operations that are supported by the Web service, as well as the request and response messages associated with each operation. In principle there are many choices for the transport protocol; in practice SOAP on top of HTTP is by far the most popular choice. Figure 3-13 shows part of the `<binding>` element for the `getIfColumn` Web Service.

```
<binding name="GetIfColumnServiceBinding"
      type="tns:GetIfColumnServiceInterface">
  <SOAP:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>

<operation name="getIfIndex">
  <SOAP:operation soapAction=""/>
  <input>
    <SOAP:body use="encoded" namespace="urn:utMon"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
    <SOAP:body use="encoded" namespace="urn:utMon"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
</operation>

<operation name="getIfDescr">
  <SOAP:operation soapAction=""/>
```

Figure 3-13: : Binding definition for getIfColumn

The <service> element

The <service> element is used to give the Web Service a name and specify its location (a URL). Figure 3-14 shows this element for the example of the `getIfRow` Web Service; the name is `GetIfRowService`, the location is `http://yourhost.com` and the transport protocol being used is defined by the `GetIfRowServiceBinding`.

```
<service name="GetIfRowService">
  <documentation>GetIfRow service</documentation>
  <endpoint name="GetIfRowService"
     binding="tns:GetIfRowServiceBinding">
    <SOAP:address location="http://yourhost.com"/>
```

Figure 3-14: : Service element definition for getIfRow

3.3.2 Performance

One of the arguments against the use of Web Services has been that the performance of Web Services for management would be inferior compared to the performance of SNMP. As will be shown in this section, this argument is invalid, however. although there are scenario's in which SNMP provides better performance, there are other scenaria in which Web Services perform better. In general, the following statements can be made:

- In case a single object should be retrieved, SNMP is more efficient than Web Services.
- In case many objects should be retrieved, Web Services may be more efficient than SNMP.
- The encoding and decoding of SNMP PDUs or WSDL messages is less expensive than retrieving management data from within the kernel. The choice between BER or XML encoding is therefore not the main factor that determines performance.
- Performance primarily depends the quality of the implementation. Good implementations perform magnitudes better than poor implementations. This holds for SNMP as well as Web Services implementations.

In literature several studies can be found on the performance of Web Services based management. Mi-Jung Choi and James Hong have published several papers in which they discussed the design of XML-SNMP gateways. As part of their research, also the performance differences between XML and SNMP based management have been investigated [18,19]. To determine bandwidth, they've measured the XML traffic at one side of the gateway, as well as the corresponding SNMP traffic at the other side. In their test set-up separate SNMP messages were generated for every object that had to be retrieved; the possibility to request multiple objects via a single SNMP message was not part of their test scenario. The authors also investigated delay and resource (CPU and memory) usage of the gateway. They concluded that, for their specific test set-up, XML performed better than SNMP.

Whereas the previous authors compared the performance of SNMP to XML, Ricardo Neisse and Lisandro Granville focussed on SNMP and Web services [20]. Just like the previous authors, they implemented a gateway and measured traffic at both sides of that gateway. Two gateway translation schemes were distinguished: protocol level and object level. In the first scheme a direct mapping exists between every SNMP and Web services message. In the second scheme a single, high level, Web services operation (like `GetIfTable`) maps on multiple SNMP messages. The authors also investigated the performance impact of using secure HTTP and `zlib` compression [21]. NuSOAP was used as web services toolkit. For protocol level translation they concluded that Web services always require substantial more bandwidth than SNMP. For object level translation they found that Web services perform better than SNMP if larger numbers of objects should be retrieved.

Another interesting study on the performance of Web services and SNMP is performed by George Pavlou et al.[22,23]. In fact their study is much broader than performance, and also includes CORBA-based approaches. They take as example the retrieval of TCP MIB variables, and measure bandwidth and round trip delay. As opposed to the previous studies, no gateways are used, but a dedicated Web Services agent was implemented, using the WASP toolkit. The performance of this agent was compared to a Net-SNMP agent. The authors did not investigate other SNMP agents nor the effect of fetching the actual management data from within the system. They concluded that Web services is a promising technology but has more overhead than SNMP.

In the remainder of this section a study performed by one of the authors of this chapter will be discussed; an extended version of this study can be found in literature [24]. For this study measurements were performed on a Pentium 800 Mhz PC, running Debian Linux. The PC was connected via a 100 Mbit Ethernet card. Four Web Services prototypes were build: one for retrieving single `ifTable` objects, one for retrieving `ifTable` rows, one for retrieving `ifTable` columns and one for retrieving the entire `ifTable` (see previous Section "Example of Web Service-Based Management"). To retrieve management data from within the system, it was decided to use the same code for the SNMP and Web Services prototypes. In this way, differences between the measurements would be caused by differences in SNMP and Web Services handling, and not by other, for this study irrelevant differences. For this reason the Web Services prototypes were incorporating parts of the well-known Net-SNMP open source package. All SNMP specific code was removed and replaced by Web Services specific code. This code was generated using the gSOAP toolkit (V2.3.8) [25,26], which turned out to be quite efficient, since it generates a dedicated skeleton and does not use generic XML parsers, such as DOM and SAX. For compression `zlib` was used [27]. The performance of the Web Services prototypes was compared to more than twenty SNMP implementations, including agents on end systems (Net-SNMP, Microsoft Windows XP agent, NuDesign and SNMP Research's CIA agent) as well as agents within routers and switches (Cisco, Cabletron, HP, IBM and Nortel).

BANDWIDTH USAGE

One of the main performance parameters is bandwidth usage. For SNMP such usage can be derived analytically, since the PDU format of SNMP is standardized. For Web Services there are no standards (yet), which means that it is not possible to analytically derive formula's that give lower and upper bounds for the bandwidth needed to retrieve `ifTable` data. In fact, the required bandwidth depends on the specific WSDL definition, which may be different from case to case. This section therefore only discusses the bandwidth requirements of the prototype that fetches the entire `ifTable` within a single interaction; the WSDL structure of that prototype has been discussed in the previous Section .

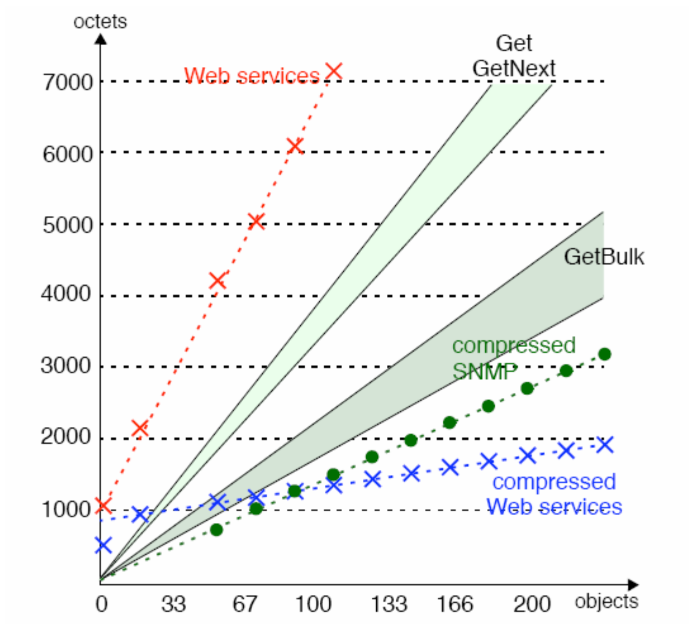


Figure 3-15: Bandwidth usage of SNMP and Web Services

Figure 3-15 shows the bandwidth requirements of SNMPv1 and Web Services as function of the size of the `ifTable`. Bandwidth usage is determined at the application (SNMP / SOAP) layer; at the IP layer SNMP requires 56 additional octets and Web Services between 273 and 715 octets, depending on the number of retrieved objects and the compression mechanism applied. In case of SNMPv3, an additional 40 to 250 octets are needed.

The figure shows that SNMP consumes far less bandwidth than Web Services, particularly in cases where only a few objects should be retrieved. But, even in cases where a large number of objects should be retrieved, SNMP remains a factor two (`get`) to four (`getBulk`) better than Web Services. The situation changes, however, if compression is used. Figure 3-15 shows that, for Web services, compression reduces bandwidth consumption with a factor 2 in cases where only a single object is retrieved; if 50 objects are retrieved the gain is a factor 4, for 250 objects it is even a factor 8. In cases where more than 70 objects should be retrieved, compressed Web services therefore perform better than SNMP. This result may be useful whenever interface data should be retrieved for hundreds of users, for example in case of access switches or DSLAMs. The figure also shows that the bandwidth consumption of compressed SNMP, as defined by the previous Evolution Of SNMP (EOS) working group, is approximately 75% less than traditional SNMP. This form of SNMP compression is known under the name ObjectID Prefix Compression (OPC).

CPU TIME

Figure 3-16 shows the amount of CPU time needed for coding (decoding plus the subsequent encoding) of SNMP (BER) and Web Services (XML) encoded messages. The figure shows that, for an SNMP message carrying a single object, the BER coding time is roughly 0.06 ms. A similar, XML encoded message, requires 0.44 ms, which is seven times more. Coding time increases if the number of objects within the message increases. Coding an SNMP message containing 216 objects requires 0.8 ms; coding a similar Web Services messages requires 2.5 ms. We may therefore conclude that XML encoding requires 3 to 7 times more CPU time than BER encoding.

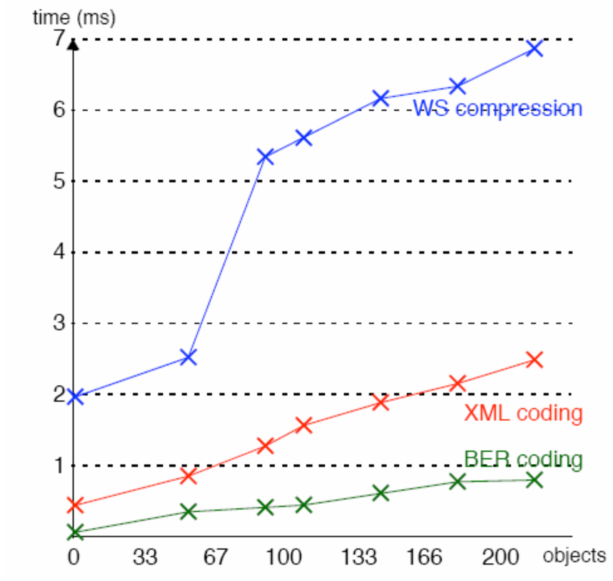


Figure 3-16: CPU time for coding and compression

The figure also shows the effect of compression: compared to XML coding, compression turns out to be quite expensive (a factor 3 to 5). In cases where bandwidth is cheap and CPU time expensive, it might therefore be better to not compress Web Services messages.

To determine how much coding contributes to the complete message handling time, also the time to retrieve the actual data from within the system was measured. In case of the `ifTable` such retrieval requires, amongst others, a system call. It turns out that retrieving data is relatively expensive; fetching the value of a single object usually takes between 1.2 and 2.0 ms. This time is considerably larger than the time needed for coding.

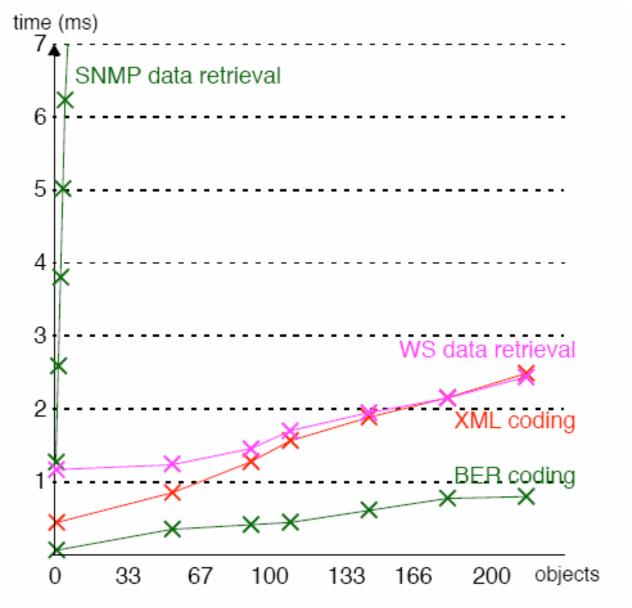


Figure 3-17: : CPU time for coding and data retrieval

The results are shown in Figure 3-17; to allow easy comparison, the BER and XML coding times are shown as well. The figure shows that data retrieval times for Net-SNMP increase quite fast; five objects already require more than 6 ms, for 54 objects 65 ms were needed and for 270 objects even 500 ms. These times can only be explained from the fact that Net- SNMP performs a new system call every time it finds a new `ifTable` object within the `Get(Bulk)` request; Net-SNMP does not implement caching. The conclusion must therefore be that data retrieval is far more expensive than BER encoding; CPU usage is not determined by the protocol handling, but by the quality of the implementation.

MEMORY USAGE

Memory is needed for holding program instructions as well as data. In case of data memory, a distinction can be made between static and dynamic memory. Static memory is allocated at the program start, and remains constant during the program's lifetime. Dynamic memory is allocated after a request is received, and released after the response is transmitted. If multiple requests arrive simultaneously, dynamic memory is allocated multiple times.

Figure 3-18 shows the memory requirements of Net-SNMP and the Web Services prototype. It turns out that Net-SNMP requires roughly three times more instruction memory than the Web Services prototype. Also Net-SNMP requires twenty to forty times more data memory, depending on the number of objects contained in the request. The importance of these numbers should not be overestimated, however, since the functionality of Net-SNMP is much richer than that of the Web Services prototype. For example, Net-SNMP supports three different protocol versions, includes encryption, authentication and access control, and is written in a platform independent way.

	instructions	data	
		static	dynamic
SNMP	1972 KB	128 KB	70 - 160 KB
Web services	580 KB	470 B	4 KB

Figure 3-18: Memory requirements

ROUND TRIP DELAY

In the previous subsections the performance of the `getIfTable` Web Service was compared to that of Net-SNMP. This subsection compares the performance of this Web Services prototype to other SNMP implementations. Since it is impossible to add to existing agents code that measures BER encoding and data retrieval times, round trip delay, as function of the number of retrieved objects, has been measured instead. This delay can be measured external to the agent and thus does not require any agent modification.

	1	22	66	270
WS	1,7	2,6	10,3	36,5
WS-Comp	3,3	4,3	5,6	11,8
SNMP-1	1,0	2,5		
SNMP-2	1,0	2,7		
SNMP-3	1,1	3,2	6,9	14,9
SNMP-4	2,0	15,2		
SNMP-5	3,7	18,3		
SNMP-6	1,3	18,9	53,0	
SNMP-7	2,9	58,0	167,0	695,0
SNMP-8	3,1	58,2	168,8	700,0
SNMP-9	3,5	62,0	183,8	767,0
SNMP-10	5,0	80,9		
SNMP-11	12,0	86,9	244,3	
SNMP-12	12,4	257,9		

Figure 3-19 : Round trip delay (in ms)

Figure 3-19 shows the results for 12 different SNMP agents, as well as the `getIfTable` Web Service prototypes. It is interesting to note that round trip delay for normal Web Services increases faster than that of compressed Web Services. For most SNMP measurements `getBulk` was used, with max-repetition values of 1, 22, 66 and 270; for agents that do not support `getBulk`, a single `get` request was used, asking for

1, 22, 66 or 270 objects. Not all agents were able to deal with large sized messages; the size of response message carrying 270 objects is such that, on an Ethernet LAN, the message must be fragmented by the IP protocol.Since the hardware for the various agents varied, the delay times shown in should be used with great care and only considered as an indication. Under slightly different conditions, delay times may be quite different. For example, the addition of a second Ethernet card may have severe impact on delay times. Also delay times will change if other objects are retrieved than `ifTable` objects. In addition, the first SNMP interaction often takes considerable more time than subsequent interactions.

Despite these remarks, the overall trend is clear. Just as with Net-SNMP, for most agents delay time heavily depends on the number of retrieved objects. It seems that several SNMP agents would benefit from some form of caching and, after more than 15 years of experience in SNMP agent implementation, there is still room for performance improvements. From a delay point of view, the choice between BER and XML encoding doesn’t seem to be of great importance. The Web Services prototype performs reasonably well and, for multiple objects, even outperforms several commercial SNMP agents.

3.4 Conclusion

The significance of Web Services for Management is recognised and they are identified as an appealing enabler of Autonomic Management.

Some of the unique features of Web Services can assist in achieving self-configuration, self-awareness and self-optimisation:

- increased performance
- platform independence and interoperability
- easy deployment on heterogeneous networks
- customisable monitoring and notifications framework
- wealth and maturity of tools and development potentials

Obviously there are great improvement margins for Web Services based management. Standardisation efforts are continuing and additions to the framework are being developed. It is evident though that there is significant potential in the adoption of Web Services for Autonomic Management.

4 Trust and Misbehaviour in Autonomic Systems

4.1 Introduction

An autonomic system is a system capable of managing itself and adapting to changes in accordance with policies and objectives defined by a human administrator. This system can interact with other autonomic systems to perform collaborative tasks. Moreover, the system itself can be composed of a distributed set of autonomous components. In these two cases, trust is a crucial issue to evaluate how an autonomic system (or an autonomic component) is reliable and if it will interact properly when it is solicited by another one. The collaborative entity can thus itself be considered a distributed autonomic system (just as an organism is a cooperative collection of autonomous cells).

The meaning of “trust” has therefore to be defined [28]. Trust is a subjective measure that an agent uses to offset risk. If an agent is going to expose itself to risk it exhibits greater trust the greater the potential risk. There is clearly a quantitative relationship to trust and risk in the most general case. This measure is context dependent due to the subjectivity of the measurement. It therefore remains to determine whether sufficient information is available to agents to determine or estimate these quantities.

The deployment of trust and reputation mechanisms in autonomic systems requires that the systems observe one another in order to determine to what extent they behave in accordance with what they have implicitly or explicitly claimed to the others (their “promises”). In addition an observing system can, in turn, be evaluated by the other systems to determine the reliability of its observations. As information about trust evaluations are made public or spread throughout a collaborative group, one can talk about the concept of reputation, or trust within a community.

This document will detail five key concepts of trust in autonomic systems. First, we will present how the autonomic systems can describe themselves and advertise their behavioural properties. Then, we will show how these properties can be monitored in an autonomic manner and to what extend analysis models are necessary to improve the efficiency of this monitoring. A third section will discuss how these autonomic systems can estimate, based on the observations, whether a system is reliable or not. Finally, we will show how trust is a matter of policy and how it relates to reputation.

These concepts will be illustrated with the case study of ad-hoc networks, although we note that all networks with finite reliability can be considered as ad hoc networks in the sense of their reliability [29]. We shall go further however in thinking of ad hoc networks as being self-managing. These autonomic networks are spontaneously formed from a set of mobile devices (laptops, PDAs, mobile phones, sensors), without requiring any pre-existing fixed infrastructures. Thus mobile devices are responsible for maintaining the network and for interacting as routers to forward packets on the behalf of others through multi-hop communications. A key issue is to provide trust and reputation mechanisms to evaluate how ad-hoc nodes are reliable and therefore identify the nodes that may deteriorate the network availability and performances.

Internal and external evaluations of systems

The notion of trust can be defined as the expectation of one autonomic system that another autonomic system will properly behave in accordance with a set of behavioural properties. These behavioural properties can for instance be described as promises

[30,28]. Thus, a first requirement is to describe the behavioural properties that can be expected by an autonomic system A from an autonomic system B. These behavioural properties will serve as a reference basis to evaluate the reliability of the autonomic system B.

We can distinguish two categories of descriptions depending on which of the two autonomic systems defines them: (1) internal descriptions that are claimed by the autonomic system B itself to advertise what it is capable of doing, and (2) external descriptions defined by the autonomic system A to define its own criteria regarding what the system should be able to do in order to be considered as reliable.

The internal description can be explicitly defined by an autonomic system using promises, or can be implicitly defined when a system provides a standardized service. In that case, it is possible to determine the system's normal behaviour with respect to this service. Typically, if we consider the scenario of ad-hoc networks, an ad-hoc node (seen here as an autonomic system) implicitly describes part of its behavioural properties when it implements a routing protocol. It is therefore easier for an external system to evaluate the normal behaviour of this node and to determine if it is not reliable based on the protocol specifications. A typical promise made by nodes forming an ad hoc network is to route packets for one another. Thus the reliability of a node in keeping this promise is one approach to measuring trustworthiness. In that context, work has been carried out at INRIA and UniS that seeks to give nodes in an ad hoc network the necessary tools to measure this trustworthiness. A node's behaviour is determined based on the promises obtained from its implicit internal description which can be learned from the routing protocol's rules once the node has accepted to abide by them.

Work at INRIA proposed in [31,32] identifies unreliable ad-hoc nodes by analysing the routing traffic generated by nodes implementing the standardized OLSR routing protocol. The optimized link state routing protocol (OLSR) [33] is a proactive routing protocol that optimizes the pure link state routing algorithms to cope with the requirements of mobile ad-hoc networks. As in a pure link state algorithm, the normal and implicit behaviour of an OLSR node can be described in two main operations: (1) the node determines the list of direct-connected neighbour nodes by periodically emitting beaconing hello messages and (2) it exchanges this link state information with the other nodes throughout the network by flooding topology control messages.

We focused on the specification of the first operation (periodic emission of beaconing messages) to determine the expected behavioural properties of nodes in terms of the perceived intermittence of their signals. Identification between regular signals and reliability is assumed, so that we may use signal intermittence to detect unreliable nodes. Although, intermittence is common to ad-hoc networks, it can be caused by two major causes.

The first one is regular intermittence (usually over longer time scales), where nodes are moving around and where direct neighbourhood relationships are observed by participating nodes. Two nodes might have a direct link-level neighbourhood relationship at one time moment, then due to mobility and user behaviour loose it and maybe re-establish it at a later time moment. The second one is pathological intermittence (which is postulated to happen over shorter time scales) that may reveal: miss-configuration of the routing protocol, malicious activities of nodes flooding the network or dropping packets supposed to be routed, errors at the physical layer, or also battery failures. The fact that an OLSR node has to periodically emit beaconing messages to the neighbourhood was considered in our scheme as an implicit description of node behavioural properties in order to detect unreliable nodes.

On the other hand, work at UniS [34, 35] proposes to identify the unreliability of a node in terms of the node's level of misbehaviour. This scheme is not tightly coupled to any specific routing protocol and therefore it can operate regardless of the routing strategy adopted. The algorithm followed by this approach first determines the number of packets that an analysed node was given for it to forward and the number of packets that the node has actually forwarded. Then both values are compared and a quantitative concept on the level of misbehaviour/unreliability exhibited by the node is yielded. Our algorithm requires all nodes to maintain a cache table with information about each node directly communicating with them. Each table entry must include the ID of the communicating node and two evaluation metrics: the number of packets sent to the node for it to forward, and the number of packets received from the node that did not originated at it (i.e. packets forwarded by the node). At a later stage an agent, i.e. a node that decides to carry out a behaviour-check on another node (the analysed node), sends out a metrics request and nodes with relevant evaluation metrics on the analysed node reply. After all evaluation metrics have been received, the agent proceeds to add them up, compare them and calculate the level of misbehaviour exhibited by the analysed node.

Deciding whether a node should be trusted or not depends directly on the level of misbehaviour exhibited by such a node and the requirements the agent has for a particular situation. Therefore, two agents may have different local appreciations on whether a node is to be trusted or not. This phenomenon rises from the fact that trust is a subjective context dependent measure, and while an agent may consider that its potential risk is low, other agent may perceive it as high and demand a higher level of local trust in order to employ other system's resources.

4.2 Monitoring and Characteristic Scales

Having announced the behavioural properties expected for an autonomic system, the second step consists in monitoring and analysing these properties. This activity is usually performed by the other autonomic systems in the group, in a distributed manner. A key issue is to avoid any biased observations that can be locally generated by an autonomic system. If false observations may be voluntary reported by malicious systems, they are more generally due to unreliable systems that monitor other autonomic systems and consider them as unreliable systems even if they are not. As we have no guarantee regarding the reliability of local observations, analysis methods are required to correlate and synthesize the observations performed by the autonomic systems at a higher scale.

Assumptions: normal and anomalous behaviour

The philosophy of error detection involves a number of assumptions. One of these concerns the time scales of the problem. If a node is considered unreliable, we mean that its behaviour is inconsistent over a given time-frame. We must specify over what time scale we expect consistency and whether we expect consistency of absolute value or only of the average. These notions are related to those of Quality of Service and Service Level Promises. In any real system, connected to its environment there are details that cannot be controlled or guaranteed precisely, thus we must specify acceptable uncertainty in behaviour.

An anomaly is defined (as a subjective policy) to be an event that is not considered to be within the acceptable margins of uncertainty. An anomaly is also called a fault. The detection of faults is related to the idea of reliability, but whereas a fault is a “short term” event, reliability is understood to be an average measure (an expectation value) of behaviour which requires “long times” to assess. One must always define the time scales “short term” and “long term” for a given behavioural process in order to make sense of measurements.

We can mention two approaches to the assessment of nodes in an autonomic network: the detection of short term faults and estimation of long term reliability.

In cfengine the autonomous monitoring approach is used to let nodes in a neighbourhood of any given point observe one another and report when they can no longer communicate with a neighbour. This “neighbourhood watch” scheme works in an entirely autonomous fashion using high level connections. Each node remembers the average interval between communications with another node and can detect when an anomalous amount of time has passed with no signal. It can then be assumed that the state of the host is anomalous, and we can decide as a matter of policy whether this is to be called a fault.

On the other hand, in UniS’s approach nodes are consistently monitoring each other’s packet forwarding behaviour allowing the network to obtain several measures on each node’s behaviour. These behaviour measurements obtained over a short term time scale do not offer a high degree of certainty on how reliable a node is since an anomalous behaviour can be observed due to other reasons rather than packet dropping owing to a node’s own fault, e.g. packet dropping due to network congestion and queue saturation. In spite of this, this anomalous behaviour is to be call a fault if the network policy so determines. Nodes in the network keep a record of these fault occurrences to allow for an evaluation that can offer a higher degree of certainty on a node’s reliability over a long term time scale.

At the longer time scale we are interested in the history of the statistical fluctuations in behaviour. These can be learned as a cumulative Bayesian process in which the probability of certain classes of behaviour is updated as time progresses.

It is important to realize that one can only approach justifiable statistical estimates of behaviour after one has monitored for a long time. There is a conflict of interest here, as ad hoc networks are expected to have highly variable behaviour. Deciding whether behaviour is normal or anomalous is therefore much harder unless one has some kind of foreknowledge about the time scales over which hosts are expected to behave consistently.

Analysis Methods

We present here two analysis methods that can be used to correlate the observations performed by a set of autonomic systems. We will consider in this modelling a distributed set noted V={v1, v2, v3,…,vn} of n autonomic systems that monitor themselves. An observation will correspond to the measurement of one behavioural property (assuming by convention that a high measurement will identify an unreliable behaviour).

i. K-means Analysis (Data Clustering)

The k-means analysis [36] consists in clustering a set of elements into k partitions based on their attributes. This classification method can be applied to our scenarios in order to separate the autonomic systems into two clusters (noted cl1 and cl2): reliable ones and unreliable ones. In that case, the set of elements to be classified stand for the set of autonomic systems, while the attributes of an element stand for the observations performed by the n autonomic systems. We can notice that the observations done by the system itself from the local point of view is included in these observations. It makes sense that an autonomic system can monitor itself and determines that it is not capable to perform what it claims.

Each autonomic system is then seen as a point vi in a n-dimensional vector space, where the j-th dimension corresponds to the observation done by the autonomic system vj. The k-means analysis will put together the autonomic systems showing similar properties by associating each point vi to the cluster cl in {cl1,cl2} whose centroid (noted centroid(cl)) is the nearest. The centroid of a cluster cl corresponds basically to the centroid of the points part of this cluster in the n-dimensional vector space. The algorithm aims to minimize the objective function defined by the following equation:

$$E = \frac{1}{2} \cdot \sum_{cl \in \{cl_1, cl_2\}} \sum_{v_i \in cl} \left\| v_i - centroid(cl) \right\|^2$$

In this equation, cl1 and cl2 are the clusters representing the two categories of systems and || vi - centroid(cl)|| is the distance between a point vi and the centroid of its cluster cl. The reliable systems are then located in the cluster with the centroid presenting the lowest value, as we considered by convention that a high measure value identifies an unreliable system.

ii. Principal Components Analysis (Data Dimension Reduction)

The Principal Components Analysis (PCA) is another powerful and common analysis method to synthesize the observations performed by a distributed set of autonomic systems. Let’s consider these observations as a multidimensional data set. This analysis method is capable of simplifying this set of observations and retaining the most important characteristics. It relies on an orthogonal linear transformation which reduces the multi-dimensional dataset to lower dimensions by extracting the characteristics that contribute the most to its variance.

The PCA analysis method provides dimension reduction, while the k-means clustering method provides data clustering. The article [37] recently proves how these two methods relate, by showing that the principal components are in fact the continuous solutions to the discrete cluster membership indicators for K-means clustering.

The first method (k-means) was considered in the framework of ad-hoc networks to detect unreliable nodes based on their intermittence. We introduced an intermittence measure calculated from information already available at the routing layer. Each network node observes its neighbour nodes by analysing the distribution of hello messages received from them. The intermittence measurements are then stored into an n-dimensional vector space, and the k-means analysis is executed on this space to determine the unreliable nodes.

4.3 Policy Decision

Monitoring and classifying autonomic systems based on their properties is a necessary condition before deciding which systems are reliable (and which ones are not). This decision can often be reduced to the choice of a threshold value. The autonomic systems are ranked or classified according to a criterion, and then the systems showing higher (or lower) values than the threshold value are considered as unreliable systems. In that case, the trust and reputation mechanism can be seen as a diagnostic test where we test if an autonomic system is reliable or not.

The decision (choice of a threshold value) will directly impact on the sensibility and specificity of the diagnostic test. The sensibility is a performance metric showing how well a test picks up true cases (unreliable systems), by defining the proportion of cases having a positive test result of all positive samples tested. The specificity defines how well it detects false cases (reliable systems), by comparing the proportion of true negatives of all the negative cases tested.

The threshold value choice leads to a trade-off between specificity and sensibility. An intuitive idea of this trade-off can be given by analysing two extreme cases in terms of threshold values. In a first extreme case, we decide that all the autonomic systems are unreliable (threshold value set to 0), then the sensibility reaches its maximal value 1 while the specificity is at 0. In a second extreme case, we decide that all the autonomic systems are reliable (threshold value set to infinity), sensitivity is at 0 while specificity equals 1.

Simulations may provide us support to determine the sensibility and specificity performances of a method and to evaluate the theoretical optimal threshold values. Typically, with the case study of ad-hoc networks, we plotted the receiver operating characteristics (ROC curves) to compare the true positive rate and the false positive rate of several collaborative methods used to identify unreliable ad-hoc nodes. These performance analyses do not replace the requirement of taking a decision, but provide some indications on the impact of a decision.

4.3.1 Trust, a matter of policy

The fact that one can only distinguish faulty behaviour as a matter of policy has a strong implication. One can only determine trust as a matter of policy. In [29] we have shown that any estimate of behaviour is subject to an arbitrary (subjective) decision about whether behaviour is acceptable or not. In statistical terms, we cannot therefore decide the «fault reliability» of a node without making a subjective judgement. The only way around this matter is to arrange for agreements between the nodes in a collective about the criteria for this judgement so that the decision is taken only in one place, or is arrived at by consensus.

Since trustworthiness is by definition an agent's reliability in keeping its behavioural promises, the same arguments apply to trust. The decision to trust another agent is essentially a policy decision. It can be motivated by quantitative estimates of reliability, but the final decision is a subjective (essentially arbitrary) judgement.

4.3.2 Reputation

A «reputation» is a propagated valuation of something (not necessarily a promise), it is received by one agent about something originating from another agent. A natural basis for reputation (and one that is used on `reputation systems' in computing) is the valuation of an agent's trustworthiness. Here we consider the effect that such transmission of information has on the local trust within a network of agents. A natural approach to defining a trustworthiness reputation (communal trust) is to use social network models to define a self-consistent measure.

Suppose that an agent T trusts an agent S to keep its promise to R with probability ET ($S \rightarrow R$), and suppose that this agent T promises to transmit this as S's reputation to another agent U. U's estimate of the trustworthiness of T's communication is its valuation of reputation is its expectation that the promised estimate is correct, i.e. U's estimation of the promised valuation of its neighbour [ref 1].

Can we say what U's expectation for the reliability of the original promise a: b → c should be? In spite of the fact that probabilities for independent events combine by multiplication, it would be presumptuous to claim that the simple multiplication of valuations arriving from different sources would yield a trustworthy value, since U does not have any direct knowledge of S's behaviour to R. So he must evaluate the trustworthiness and reliability of the source as well.

Although we can define pathways by rules of probability to give mathematically defined values, the semantics of the result are unclear. It is important to see that no agent is obliged to make such a policy to faithfully transmit a reputation. Thus trust and reputation do not propagate in a faithfully recursive manner. There is, moreover, in the absence of complete and accurate common knowledge by all agents, an impossibility of eliminating the unknowns in defining the expectation values.

In [34] we have provided definitions that are based on fair-weighted computations of reliability to an impartial observer. These can be compared to common trust models such as peer-trust and Trusted Third Parties. If we consider our case study, we can note that any data shared in the evaluation of an ad-hoc node is in fact a reputation. Any direct observation requires an element of trust.

4.4 Misbehaviour in Autonomic Systems

The concept of misbehaviour is similar in many ways to that of trust introduced in section 1. However, while trust is a subjective measure that an agent uses to evaluate a potential risk based on an autonomic system's reliability, misbehaviour is a subjective measure to evaluate whether an autonomic component/system is causing disruption intentionally or not. Thus, misbehaviour is a metric that can be computed with other metrics to give an indication of the overall trust of a system.

Autonomic components/systems may exhibit anomalous behaviour for various reasons. For example, a component may try to hinder a system's performance by not obeying globally acceptable rules (implicit descriptions) or by not fulfilling its promises (explicit descriptions), which is a clear case of misbehaviour. However, some systems can behave erratically without intending to do so. For instance, a well behaved system whose functionality has been compromised by a misbehaving component, in this case some of the tasks carried out by the system rely on the promises made by the misbehaving component, since the misbehaving component does not fulfil its promises the system as a whole cannot fulfil its own promises, thus exhibiting anomalous behaviour. The solution in this case is to isolate the component rather than penalising the whole system which could prove to be useful if all its components behave adequately.

Misbehaviour, as trust, is a context dependant measure and it is necessary to determine whether agents have available sufficient information to compute and yield reliable measurements for each specific environment. Additionally, determining misbehaviour also requires systems and components to monitor each other's behaviour constantly to ensure they behave in accordance to their implicit and explicit descriptions (section 2).

The remainder of this section will illustrate the concept of misbehaviour and its adaptation to autonomic systems with a case study based on the principle of conservation of flow applied to mobile ad hoc networks (MANETs). Providing self-managing MANETs with misbehaviour detection tools is imperative due to their inherent weaknesses as a result of the broadcasting nature of the wireless channel, which makes them vulnerable to a wide variety of attacks launched by misbehaving nodes.

An approach based on the principle of conservation of flow (PCF)

One of the basic principles on which a mobile ad hoc network's functionality is based states that nodes in a network cooperate by forwarding packets on behalf of each other when destinations are out of their direct wireless transmission range. Work at UniS [34,35] has focused on finding ways to protect the packet forwarding functionality of routing protocols against nodes that purposefully drop packets in order to hinder the overall network performance. In this section, an approach that takes advantage of the principle of conservation of flow to detect misbehaving nodes is described [38].

Introducing the principle of conservation of flow

We now formally introduce the principle of flow conservation over an ideal static network model:

- Let v_j be a node such that $v_j \in V$, where $V = \{v_1, v_2, v_3 \ldots v_N\}$ is the set of all nodes in the network, N is the total number of nodes in the network, and $j = 1, 2, 3 \ldots N$.
- Let U_j be the subset of nodes in the network which are neighbours of v_j, i.e. U_j is the neighbourhood of v_j. It follows that $v_j \not\in U_j$ and also $U_j \subset V$.
- Let Δt be the period of time elapsed between two points in time t_0 and t_1 such that $\Delta t = t_1 - t_0$.
- Let T_{ij} be the number of packets that node v_i has successfully sent to node v_j for v_j to forward to a further node; $v_i \in U_j$, $v_j \in U_i$, $i \neq j$ and $T_{ij}(t_0) = 0$.
- Let R_{ij} be the number of packets that node v_i has successfully received from node v_j that did not originate at v_j; $v_i \in U_j$, $v_j \in U_i$, $i \neq j$ and $R_{ij}(t_0) = 0$.

If all nodes $v_j \in V$ remain static for a period of time Δt during which no collisions occur in any of the transmissions over an ideal (noiseless) wireless channel, then for a node v_j:

$$\sum_{\forall i | v_i \in U_j} R_{ij}(t_1) = \sum_{\forall i | v_i \in U_j} T_{ij}(t_1) \qquad (1)$$

Equation (1) states the fundamental premise of the flow conservation principle in an ideal static network applied to packets rather than raw bytes. It states that if all neighbours of a node v_j are queried for i) the amount of packets sent to v_j to forward and ii) the amount of packets forwarded by v_j to them, the total amount of packets sent to and received from v_j must be equal.

Adapting the principle of conservation of flow to MANETs

In practice networks exhibit conditions that are far from ideal. First of all, the wireless channel is error prone and packets get lost while in transit. Secondly, collisions happen when the network uses protocols where nodes have to compete for the medium, such as when the link layer protocol is based on the distributed coordination function (contention period) of the IEEE 802.11 a/b standard. On the other hand, an overloaded node may temporarily lack the CPU cycles, buffer space or bandwidth to forward packets. In addition, some reactive routing protocols, e.g. AODV, cause buffered packets to be dropped if they go through a path that is even temporarily unavailable. Thus, a node may exhibit anomalous behaviour even if it is not purposefully doing so.

For these reasons equation (1) cannot be applied in a rigorous manner and a threshold needs to be established to account for packets dropped by a node through no fault of its own. Equation (2), which holds for well behaved nodes, reflects this change:

$$(1-\alpha_{threshold}) \sum_{\forall i | v_i \in U_j} R_{ij}(t_1) \le \sum_{\forall i | v_i \in U_j} T_{ij}(t_1) \qquad (2)$$

The $\alpha_{threshold}$ factor can take values between 0 and 1 and as we shall see plays an important role in the detection power of our proposed algorithm, i.e. the capability of the algorithm to detect misbehaving nodes. The lower $\alpha_{threshold}$ is the more likely it is that our algorithm detects any anomalous behaviour. However, it also means that the probability of a false detection increases. A false detection occurs when the result of a single evaluation of a node mistakenly determines that the node appears to be misbehaving. Therefore, fine tuning is required to reach a fair point in this trade-off between specificity and sensibility

Tracking down mobile nodes to examine their behaviour

The algorithm used to track down mobile nodes allows for the collection of information on the analysed node (the node whose behaviour is under evaluation) regardless of whether it is static or on the move. This algorithm consists of a series of phases which can be performed in parallel for different analysed nodes. These phases are:

Monitoring: a node v_i keeps track of both metrics R_{ij} and T_{ij} for each node that communicates directly with it. These values are maintained in a cache table for their later retrieval when a behaviour check takes place.

Behaviour check scheduling: When a node overhears a transmission it caches the address of the transmitter, and schedules a behaviour check on this transmitting node in a period of time uniformly selected between T_{MIN} and T_{MAX}.

Behaviour check initiation: Whenever a behaviour check is triggered in a node it broadcast a metrics request packet (MREQ) including the address of the analysed node so that other nodes may reply with the corresponding metrics R_{ij} and T_{ij}. In this section a node that analyses anther node's behaviour is also called an agent.

Metrics request handling: A node that receives a MREQ checks whether the address of the analysed node appears in its cache table, if not it ignores the packet and drops it, otherwise the node continues broadcasting the packet. Any nodes having relevant metrics about the analysed node send them back in a metrics reply packet (MREP) towards the node that originally started the behaviour check, i.e. the agent.

Metrics reply handling: After initiating a behaviour check an agent waits for a preset amount of time which is long enough to allow all nodes with relevant metrics to reply back. Then it adds up the packets forwarded by the analysed node (R_{ij}) and compares them to those the node was expected to forward *(T_{ij})* using equation (2). If equation (2) holds, the node is considered to be a well behaved node, otherwise the node is considered to be misbehaving. This latter case is called a detection.

Misbehaviour detection and collaborative accusation

It is important to distinguish between a detection and an accusation. A detection means that a node has identified another node that appears to be misbehaving. In contrast, an accusation occurs when a node reports another node as misbehaving. From these definitions it follows that while a detection may affect an agent’s trust in other node, an accusation will also affect the reputation) of the misbehaving node.

As stated previously reputation is a propagated valuation of something; in this case study the propagated value is an accusation on the anomalous behaviour exhibited by a node. There are therefore two issues to be tackled: i) how an agent can be confident on its measure so as not to falsely accuse a node from misbehaviour, and ii) how an agent receiving an accusation can be sure that such information is reliable. Increasing an agent’s confidence in an accusation it is about to propagate, can be done through collaborative consensus mechanisms. Thus, an agent does not accuse a node based on a detection or set of detections performed by itself, but instead it collects information from other agents that have also evaluated the behaviour of the analysed node. Then, to avoid any biased detections that can be locally generated, an agent can process the data using any of the analysis methods presented in previous sections, i.e. the k-mean analysis (data clustering) or the principal component analysis (data dimension reduction).

On the other hand, an agent that receives an accusation can be confident that it is reliable if the accusation is mutually signed by the parties involved in the accusation process, i.e. the agents providing information on the analysed node. Yielding signatures that are valid only when a determined number of parties have contributed with their secret share requires the use of cryptographic techniques such as threshold secret sharing and secret share updates [39,40,41]. These techniques provide confidence to an agent that receives an accusation by guaranteeing that such accusation was made through a collaborative consensus instead of being produced by a malicious or biased agent.

4.5 Conclusions

Trust and misbehaviour are major issues for designing and developing the models and architectures of efficient autonomic systems. We detailed six underlying key concepts and showed how they can be applied to the case study of ad-hoc networks. First, the autonomic systems are required to describe themselves to determine their behavioural properties. These descriptions can be both internal when an autonomic system specifies its own properties using promises, and external when an autonomic system specifies the properties it expects from another system. Then, the systems monitor themselves to determine if their behaviours are normal or anomalous. The goal is to detect faults at a short term and estimate reliability of systems at a longer term. Two analysis methods were proposed to synthesize the data set of observations measured by the systems. The *k*-means analysis is capable of clustering this data set to separate the reliable systems from the unreliable ones. The principal components analysis is capable of simplifying the data set by retaining the most significant characteristics. Based on these analyses, the final decision to trust other autonomic systems is essentially a policy decision. The valuation of a system’s trustworthiness can then be propagated among the other systems in the form of reputation. However, in both cases trust and reputation, the semantics of the final value are a matter of the policy of the final recipient. Finally, we presented how misbehaviour can influence trust and reputation measurements in autonomic systems/components and examined a case study based on the principle of conservation of flow.

5 Context Awareness and Network Patterns

5.1 Introduction

Communication networks nowadays tend to become a common utility, used by more and more people on everyday basis in an increasingly complex way. As people get more accustomed to the availability of communication services, the importance of mobile communication networks increases. Basic properties of mobile networks make it a suitable infrastructure for providing more “intelligent” services than in case of standard fixed environments. User may expect its terminal device to perform differently depending on various properties of the surroundings, adapting itself to the varying conditions and providing the user with best possible experience under current circumstances. In order to meet such requirements, both end-user devices and the network must be capable of gathering and processing information describing the operational environment and enabling the services to make use of such information.

This kind of information, a “context” may be useful in dynamic configuration of network parameters, as defined in autonomic management paradigm, but it may also be of much use in case of high-level applications, such as, for instance, travel route planning for mobile subscribers, gas station finders, weather forecast services etc. A travel route planning application would require the network to provide the device with detailed information about, for instance, surrounding road infrastructure state (like information about traffic jams), and that would require detailed information about subscriber's position. Optimal route would then be selected, possibly taking into consideration weather conditions on the candidate route. A gas station finder would work in similar way, requiring information about current position of the mobile terminal and perhaps gas prices at stations belonging to different vendors. These basic examples demonstrate the vast customization possibilities that context awareness brings to the mobile network users and providers.

In addition, context information can be combined with the concepts of patterns and can be an important part of distributed computing and processing with cfengine. Patterns are dissemination and aggregation algorithms that bridge the worlds of centralized monitoring and fully distributed monitoring. Patterns are motivated by the need for scalable dissemination and aggregation of data in a network. They were designed primarily with routing nodes in mind. However, the autonomic computing scenario is a tantalizing application for this kind of algorithm. Cfengine is an open source software system for autonomic host management. A cfengine host is by default a completely autonomous entity with no obligations towards other agents in a physical network. Every node is therefore individual and is not part of a pattern a priori. To use patterns as a form of inter-peer collaboration, we must therefore encode them as policy rules. So, instead of thinking of patterns as being overlays to a physical network one can, for instance, think of them as algorithms for balancing (spreading) processing across an array of agents in a logical overlay network.

5.2 Context Awareness in Autonomic Management

In order to use the wide range of information about current environment properties a flexible processing approach is required that would be capable of adapting itself to various context information models, as different service providers may choose to offer context information in different formats and of different types – yet all of this information needs to be usable with a wide range of different end-user devices, also for roaming subscribers. A comprehensive study regarding context modeling can be found in [42]. This creates a need for a translation layer that would enable different context providers to work with various types of devices. Such a solution was developed by LMU and UniBW in a form of CoCo Infrastructure that operates on Context Meta-Model instead of intermediate context information, which makes it a flexible solution to use in heterogeneous environment of different network providers (and thus context providers). The Context Meta-Model itself is the foundation on which developers may build their own specific context models in a way that can be understood and processed by CoCo Infrastructure later. The Meta-Model defines a set of context information classes and relations between them to set basic rules for context information processing.

5.2.1 The CoCo Infrastructure

The CoCo Infrastructure, like much other architectures for context-aware systems, follows an infrastructure-centered distributed services model based on client-server dialog [43]. It acts as a broker between context-aware services (CAS) that request context information and context information services (CIS) that provide context information. It therefore relieves the context-aware services from the burden of discovering context information services, data transformation, or derivation of high-level context information from low-level context information.

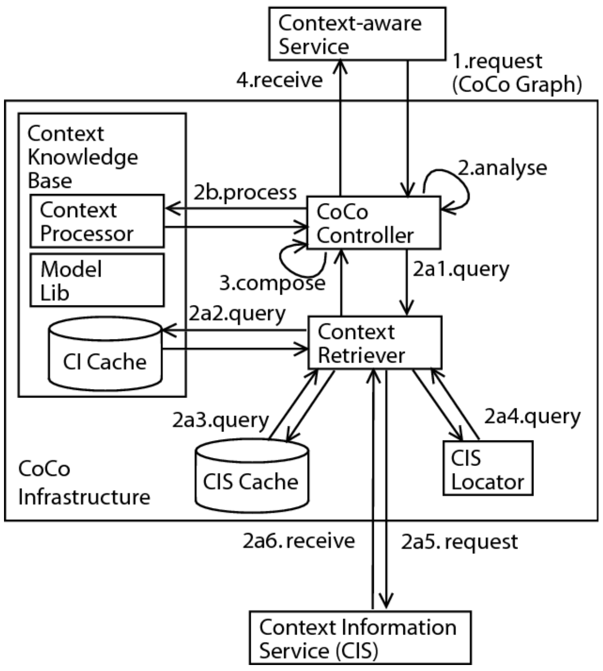


Figure 5-1: The CoCo Infrastructure

CoCo stands for Composing Context. The CoCo Infrastructure does not only support the request for a single piece of context information (like 'the current position of Alex') but also the request for composed context information (like 'the temperature at the place where Alex currently is'). This requires the ability to describe such compositions. Therefore, a language has been developed that describes the request for composed context information in so-called CoCo graphs [44]. The graph-like structures (see fig. 5-2) are expressed in XML.

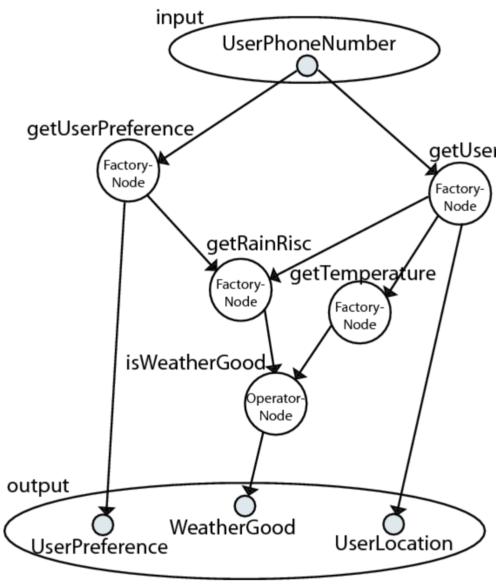


Figure 5-2: A CoCo graph describing the composition of context information

Basically a CoCo Graph is made out of two types of nodes: Factory Nodes describing the requested piece of context information and Operator Nodes containing instructions how to process one or several pieces of context information. An Operator Node may for instance be used to adapt formats of context information, or to select or aggregate information. External operations can be accessed via the address of the service providing them. Every time a node in the graph delivers its result to more than one successor, this is a branching point in the graph where succeeding nodes are processed concurrently. A synchronisation point is reached whenever a node requires input from two or more predecessors that run simultaneously previously.

The whole processing of context information from sensing to usage is shown in fig. 5-1: (step 1) First, the request in form of a CoCo graph, is sent from the context-aware service to the CoCo infrastructure where it is (step 2) analyzed by the CoCo Controller. The CoCo Controller is responsible for the parallel or sequential handling of instructions, the flow of data between nodes, and the possible reconfiguration of a graph. For every Factory Node it sends (step 2a1) a request for context information to the Context Retriever, which at first queries the Context Cache (step 2a2), whether it already has got the requested information. In case it is not available there, the CIS Cache is asked whether it already knows an appropriate context information service to retrieve the information from (step 2a3). If not it then instructs the CIS Locator to find appropriate context information services (step 2a4), to which the retriever sends related context requests. After having received context information for each request (step 2a5 and step 2a6) the retriever matches this information against the request of the controller, selects the most appropriate piece of context information and returns it to the controller.

For each Operator Node, e.g. isWeatherGood in fig. 5-2, the controller instructs the Context Processor to execute its operation (step 2b), e.g. adaptation, selection or aggregation. Context Processor and Context Cache are part of the Context Knowledge Base, which also includes the Model Lib. The Model Lib contains the ontology-based information model the middleware is based upon (for more details on the information model used, see [45]). A comprehensive description of the integration of the context meta model into the CoCo infrastructure can be found in [46].

5.3 Network Patterns and Cfengine Context Dependence

Patterns are dissemination and aggregation algorithms that bridge the worlds of centralized monitoring and fully distributed monitoring. The extreme cases are chain and star networks. Trees are structures that bridge these two extremes. We can characterize patterns by their depth and breadth. A chain has maximum depth (only one node at each distance from the source) and a star has maximum breadth (all nodes are only one hop from the source).

Here we consider only two patterns: Echo and Gap and consider how these can be implemented in cfengine using existing context awareness within the system.

Echo

In the echo pattern, a signal initiated from a single source spreads epidemically through a bounded network using a diverging spanning tree to relay a requests. When the signal reaches the edge of the network, actual responses are passed back along the tree to each parent node, where they are aggregated before being passed up to the next level. The echo pattern therefore forms a wave, spreading out from an epicentre to the edge of the network and back, collecting data as it goes. Echo is intrinsically a "push" protocol.

Gap

In the Generic Aggregation Protocol, we assume the existence of a spanning tree. Data are then passed from the leaves up towards a central point which acts as a sink for the data. In some ways, this behaviour is like the return phase of the echo pattern.

The topology manager: A key feature of the patterns above is the algorithm by which the topology of the spanning tree is decided. The GAP algorithm, as described by Stadler et al incorporates the topology adjustment mechanism into the GAP aggregation algorithm, hence combining these features into a robust protocol. However, they can be separated also. The GoCast algorithm finds such a spanning tree, for example. At this stage of the work we shall not attempt to encode automated topology management, as this requires additional subsystems. Rather we consider how patterns can be used at the logical level for distributed load balancing, using existing mechanisms within cfengine.

Cfengine and patterns

Patterns are motivated by the need for scalable dissemination and aggregation of data in a network. They were designed primarily with routing nodes in mind. However, the autonomic computing scenario is a tantalizing application for this kind of algorithm. Cfengine is an open source software system for autonomic host management. A cfengine host is by default a completely autonomous entity with no obligations towards other agents in a physical network.

Every node is therefore individual and is not part of a pattern a priori. To use patterns as a form of inter-peer collaboration, we must therefore encode them as policy rules. So, instead of thinking of patterns as being overlays to a physical network one can, for instance, think of them as algorithms for balancing (spreading) processing across an array of agents in a logical overlay network.

There are several questions to be answered about this:

1) Is the underlying physical network topology important in building a logical load sharing topology?

2) How will the topology be decided?

3) How will the topology respond to the failure of nodes?

We are currently undertaking detailed experimental studies to determine the answers to these questions. These studies are expected to be completed in July.

Promise agreements and voluntary cooperation

Promise theory is a way of modelling networks of agents cooperating in an ad hoc fashion. Cfengine can be viewed as a reference implementation of the abstract promise-theoretic scenario. (Promise theory was introduced precisely as a modelling framework that could describe cfengine, where others could not.)

The key features of promise theory are

1) There exists a collection of independent agents.

2) Agents are autonomous. They make their own decisions and no agent can be forced to perform any function by another. (Actions and decisions made by an agent are entirely voluntary.) i.e. they collaborate only as a matter of individual policy.

3) Each agent has private knowledge. It can share its knowledge with another agent by promising to reveal it (in the manner of a service).

4) There are two kinds of promises called + and -, or "service" and "use" promises. A + promise is a promise to give or provide some kind of abstract service. A - service is a promise to make use of an abstract service (if available)

Promise theory allows us to see the relationship between network patterns and policy for autonomous agents. Each arrow in the promise graph attaches to a rule in the policy to either grant access to data or to fetch available data. In this way we can build dissemination processes over graphs using node location data or context sensitivity information.

A common mistake is to think of promises as communication transactions, rather than as abstract behavioural specifiers. A promise says nothing about the details of what is communicated between agents.

We write a promise as a directed graph from one agent to another.

```
       b
 A_1  ------------->   A_2
```

b is called the promise "body" and contains details of what is being promised. An agent can only promise something about its own behaviour, not about a third party.

A reliable binding between two hosts requires both a promise to serve and a promise to use the service.

```
            +b
   A_1 -------------> A_2
            -b
   A_1 <------------  A_2
```

Using context awareness for form patterns

A method in cfengine is like a pair of promises, provided it is voluntarily declared by both parties. An MD5 hash is used to verify that the methods are in fact the same when sharing data between the parties.

The first (service) promise identifies the function being performed, as the body b(). The class expression A_1:: says that this rule applies to the context of agent A_1, which is the service provider (server host). The server=A_1 attribute matches the context expression and, from this, the agent deduces that it is the provider.

```
methods:
  A_1::
     b(params) server=A_1
             +b
   A_1 -------------> A_2
             -b
   A_1 <------------  A_2
```

The second part applied to agent A_2 and has the form:

```
methods:
  A_2::
     b(params) server=A_1
             +b
   A_1 -------------> A_2
             -b
   A_1 <------------  A_2
```

This identifies the function being performed and signals to A_2 that it will use the results performed by server A_1. Since this is not its own identity, this implies that the result is a use-promise.

If we assume that two agents use an identical configuration specification, then a remote procedure call binding can then be written

```
methods:

  A_1|A_2::

     b(params) server=A_1
```

The same text either in both contexts and a single link in a logical overlay network is added.

Promise Chains (forwarding)

```
#############################################
#
# CHAIN 6 machines 1,2,3,4,5,6
# (promise chain)
#
#############################################

classes:

 always = ( any )

 leaf   = ( node6 )
 root   = ( node1 )

#############################################
```

```
control:
  workfile   = ( "/tmp/chain-pattern" )

#############################################
methods:
  #
  # Pattern has to be coded in classes (from)
  # and servers (to)
node1|node2::
# U(p) | p - binding

  Aggregate("$(workfile)")
     server=node2
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link
```

```
node2|node3::
  Aggregate("$(workfile)")
     server=node3
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

node3|node4::
  Aggregate("$(workfile)")
     server=node4
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link


node4|node5::
  Aggregate("$(workfile)")
     server=node5
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

node5|node6::
  Aggregate("$(workfile)")
     server=node6
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

#############################################
editfiles:

   !leaf::
   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(Aggregate.ret)"
   # Handle errors so no strange loops
   ReplaceAll "Aggregate.ret" With "FAILED"
   }

  leaf::
   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(value_loadavg)"
   }

#############################################

alerts:

  root.Aggregate_chain_link::

   "Chain aggregate
$(n)$(host)=$(value_loadavg) at $(date)
$(Aggregate.ret) "
```

A simplification of the above for management ease is presented below:

```
#############################################
#
# Netlab config
#
#############################################

classes:

 leaf   = ( netlab4 )
 root   = ( netlab1 )

#############################################

control:

  workfile   = ( "/tmp/chain-pattern" )
  tempfile   = ( "/tmp/chain-temp" )

netlab1::

  serve = ( netlab3 )

netlab3::

  serve = ( netlab4 )


#############################################

tidy:

#############################################

copy:

!leaf::

  $(workfile)

    dest=$(tempfile)
    server=$(serve)
    type=checksum
    define=success
    elsedefine=failure


#############################################

editfiles:

  success::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   InsertFile "$(tempfile)"
   AppendIfNoSuchLine "copy-chain
$(host)=$(value_loadavg) at $(date)"
   }

  failure::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain - no
response from $(serve)"
   AppendIfNoSuchLine "copy-chain
$(host)=$(value_loadavg) at $(date)"
   }

  leaf::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain
$(host)=$(value_loadavg) at $(date)"
   }

#############################################
```

```
alerts:

  success::

   "Chain update succeeded"
```

```
    PrintFile("$(workfile)","6")

  failure::

    "No Chain update at $(date)"
```

Promise Trees (aggregation)

The first example of a 2 into 1 aggregation:

```
#############################################
#
# Netlab config - depth aggregation
# (promise tree)
#
#############################################

classes:

 leaf         = ( netlab3 netlab4 )
 aggregator   = ( netlab1 )

#############################################

control:

  workfile   = ( "/tmp/chain-pattern" )

  children = (
             A(netlab1,"netlab3,netlab4")
             A(netlab3,"netlab3,netlab4")
# This completes contract
             A(netlab4,"netlab3,netlab4")
             )

#############################################

methods:
  # Pattern has to be coded in
  #classes (from) and servers (to)

netlab1|netlab3|netlab4::
# U(p) | p - binding

  Aggregate("$(workfile)")
     server=$(children[$(host)])
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link
```

```
#############################################

editfiles:

   aggregator::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(Aggregate_1.ret)"
   AppendIfNoSuchLine "$(Aggregate_2.ret)"
   # Handle errors so no strange loops
   ReplaceAll "Aggregate.*ret" With "FAILED"
   }

  leaf::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(average_loadavg)"
   }

#############################################

alerts:


aggregator.(Aggregate_1_chain_link|Aggregate_
2_chain_link)::

   "Chain aggregate
$(n)$(host)=$(average_loadavg) at $(date)
$(Aggregate_1.ret) $(Aggregate_2.ret) "
```

An approximation to this with greater trust allows us to reduce the management overhead somewhat

```
#############################################
#
# Netlab config –
# breadth aggregation by trusting pull
#
#############################################

classes:
 leaf   = ( netlab4 netlab3 )
 root   = ( netlab1 )

#############################################

control:
  Split      = ( , )
  workfile   = ( "/tmp/chain-pattern" )
  tempfile   = ( "/tmp/chain-temp" )

 #                                 1
```

```
 # One link in a binary tree      / \
 # aggregation                   3   4

netlab1::

  serve = ( "netlab3,netlab4" )

#############################################

copy:

!leaf::

  $(workfile)
    dest=$(tempfile)_$(this)
    server=$(serve)
    type=checksum

    define=success
    elsedefine=failure

#############################################

editfiles:

  success::
   { $(workfile)
   AutoCreate
   EmptyEntireFilePlease
   InsertFile "$(tempfile)_$(serve)"
   AppendIfNoSuchLine "copy-chain
$(host)=$(value_loadavg) at $(date)"
   }
  failure::
   { $(workfile)
   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain - no
response from $(serve)"
   AppendIfNoSuchLine "copy-chain
$(host)=$(value_loadavg) at $(date)"
   }
```

```
  leaf::
   { $(workfile)
   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain
$(host)=$(value_loadavg) at $(date)"
   }

#############################################

alerts:

  success::

   "Chain update succeeded"
   PrintFile("$(workfile)","6")

  failure::

   "No Chain update at $(date)"
```

The Echo Pattern

Cfengine's main modus operandi is to "pull" data rather than to push. This is a natural side effect of its philosophy of voluntary cooperation. Push is disallowed. There is one exception however. We are allowed to send a single invitation to each peer to execute its existing policy using the command cfrun. The host is free to disregard this message, but for cooperation purposes it is normal for the peer to respond to such an invitation by executing its policy compliance-checking agent. We can use this mechanism to start an echo avalanche, with a pre-arranged pattern.

The start host executes cfrun to a number of "children". Each child then voluntarily executes cfengine, which in turn encapsulates the execution of cfrun directed at another set of children, which encapsulates cfrun to another set, and so on. Since cfengine aggregates the data from encapsulated processes automatically, it automatically aggregates the entire tree in a synchronized manner. This is the simplest implementation of echo which uses context sensitivity to identify parent-child relationships.

The use-promises are encoded as follows:

```
control:
       actionsequence = (shellcommands tidy)
       domain         = ( cftestnet )
       IfElapsed      = ( 1 )
       TrustKeysFrom  = ( 10.0.0 )

       node1::
              serve = ( node2:node3:node4 )

       node2::
              serve = ( node5:node6:node7 )

       node3::
              serve = ( node8:node9:node10 )

       node4::
              serve =( node11:node12:node13)

       node5::
              serve =( node14:node15:node16)

       node8::
              serve = (node17:node18:node19)

       node11::
              serve = ( node20 )
```

```
classes:
       HasChildren = ( IsDefined(serve) )

shellcommands:
       any::
              "/bin/echo $(hostname)"

       HasChildren::
          "/usr/local/sbin/cfrun $(serve)
2>&1 > /tmp/echorun.$$" background=true
          "/usr/bin/pgrep cfrun > /dev/null;
while [ $? = 0 ]; do pgrep cfrun > /dev/null;
done"
          "/bin/cat /tmp/echorun.*"

tidy:
       HasChildren::
              /tmp pattern=echorun.* age=0
```

A sample output execution is shown below illustrates how the difference hosts are visited and the sequence of actions proceeds:

```
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node2 ] ..........
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node6 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node6 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node6 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node6 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node7 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node7 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node6 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node6 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node7 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun node7 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep cfrun > /dev/null;
done...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep
cfrun > /dev/null; done
```

5.4 Future work

As part of the future work on Context Awareness, we will continue to investigate the current state-of-the art in context information management and processing, more specifically architectures for context provisioning. We will point out the weak points and good sides of already existing solutions in order to create a set of requirements that the most appropriate context management system should meet.

In addition, tests are proceeding as to the most appropriate way for deciding a topology in a cfengine peer network. The criteria for this are somewhat different to the original motivation for patterns. Once these tests have resulted in a basic understanding of the problem, we expect to add topology automation to the tool. The syntax of cfengine's voluntary cooperation model is based on peer to peer interactions, just like promise theory. This results in clumsy and cumbersome policy files for encoding patterns. Further work is expected to be able to enable regular expressions of some form to more efficiently encode the bilateral promises required for pattern policies.

6 Inspiration domains for systems with self-management capabilities

This chapter argues that we can expedite progress in systems with self-management capabilities (management systems, assistance systems, support systems) in IT management by getting inspired by existing knowledge in IT management itself or other domains. After introducing a model for the knowledge in and across domains it classifies ways of inspiration and lists promising inspiration domains.

6.1 An abstract model for scientific domains

Fig. 6-1 shows an abstract model for all domains of science where artefacts are created. In this model there are four knowledge layers:

Artefacts: This layer covers the knowledge associated with artefacts, i.e. human-made objects within this domain.

Applied research: This layer adds domain-specific knowledge that deals with the creation of artefacts and the application of models.

Domain-specific theory: Each domain adds its own layer of additional theoretic foundations which – so far – seems less applicable to other domains.

Basic nature theory: All domains share this knowledge which comprises of the common core of science. It includes basic laws of nature, mathematics and logics.

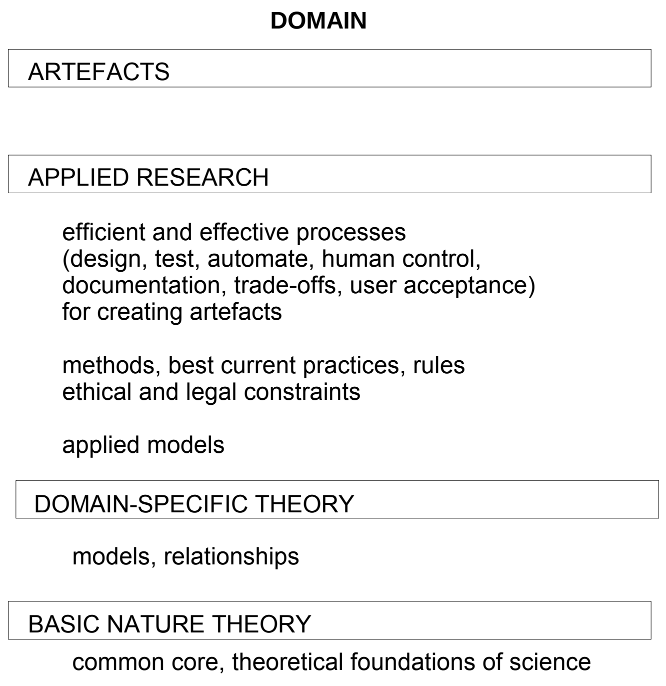

DOMAIN
ARTEFACTS
APPLIED RESEARCH
efficient and effective processes
(design, test, automate, human control,
documentation, trade-offs, user acceptance)
for creating artefacts
methods, best current practices, rules
ethical and legal constraints
applied models
DOMAIN-SPECIFIC THEORY
models, relationships
BASIC NATURE THEORY
common core, theoretical foundations of science

Figure 6-1: An abstract model for knowledge in scientific domains.

Motivating the use of Systems with Self-Management Capabilities in IT Management

The current situation in IT management shows quite a lot of deficiencies that are mostly related to the number and complexity of management operations overloading many system and network administrators. The current situation and goals in IT management make Systems with Self-Management Capabilities (SwSMC) a promising approach in this domain

Deficiencies in current IT management	**Goals in IT management**
TCO of IT solutions is dominated by increasing management efforts and costs.	Slow down the rate of increase in management efforts and costs, up to a reduction.
System administrators are faced with many details of the managed systems. They have to remember and comprehend a lot of dependencies and relationships in their complex infrastructures. Due to time constraints, lack of a deeper understanding, and lack of training, they make errors from time to time.	System enables human to give management commands at a higher level of abstraction than before. Dependencies among subsystems and constraints are taken into account by a management or assistance system.
System administrators cannot typically foresee the full effects of their actions.	System predicts effects of operator actions and warns in case of inconsistencies and predicted faults.
Too many *management* interactions are *necessary* to achieve certain goals.	System should work towards reducing the frequency for necessary management commands to achieve a certain goal. Care should be taken that the reduction of *necessary management* interactions does not necessarily imply the reduction of *potential management* interactions.
Too many interactions are *necessary* between humans about the system. This causes rising attention on the system, falling attention on the job/mission.	System should be designed in a way so that it reduces *necessary* human-human communication about the system.
System administrators still do many daily tasks in configuration, fault, performance, security management manually.	Automate frequent operations and standard tasks with self-configuration, self-healing, self-optimization, self-protection.

Table 6-1: The current situation and goals in IT management

To approach these goals we argue in favour of implementing management and assistance routines into the managed system itself and/or associated management systems (both of which we see as “self-management”). Better IT management processes and better training of IT administrators – while worthwhile and highly recommended – would have to be implemented at each service provider individually, which is an enormous organisational effort. In contrast, once designed and implemented by a vendor, SwSMC can be rolled out to many service providers.

Improvements needed in IT management

When we map the goals within the domain of IT management (see Table 6-1) to an instance of the abstract domain model (see Figure 6-1) for IT management, we can profit from improvements at three layers:

a) **Improvements in artefacts:** We want better systems with self-management capabilities. These are operations support systems, assistance systems, management systems all of which enable a certain amount of self-management.

b) **Improvements in applied research:** We want better processes to build these artefacts. Only sound processes of requirements analysis, modelling, design, implementation, testing, risk analysis and standardisation will lead to artefacts that stand up to the expectations of users.

c) **Improvements in domain-specific theory:** IT management still lacks a sound theoretical foundation for many of the investigated problems. We have to come up with better models of management processes and managed entities. Sophisticated models according to common standards would enable the use of simulations as it is now common in many other scientific domains to predict the behaviour of objects in the real world.

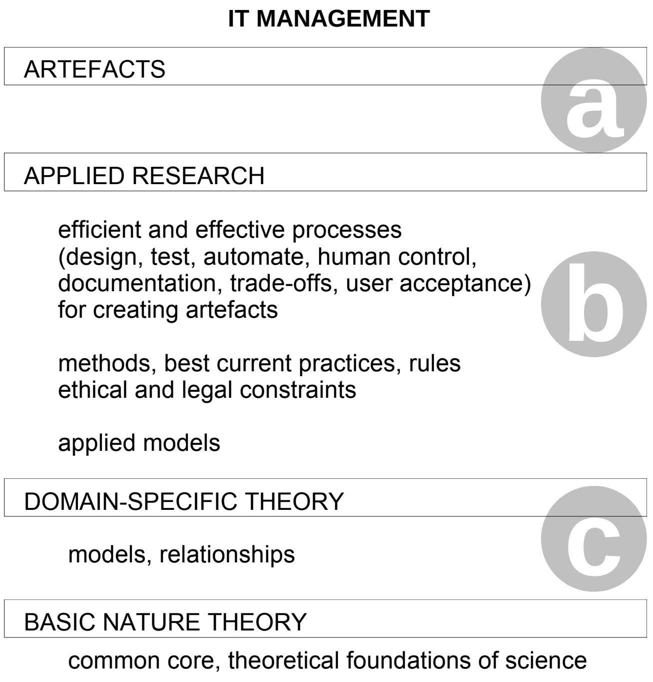

IT MANAGEMENT
ARTEFACTS
a
APPLIED RESEARCH
efficient and effective processes (design, test, automate, human control, documentation, trade-offs, user acceptance) for creating artefacts
b
methods, best current practices, rules ethical and legal constraints
applied models
DOMAIN-SPECIFIC THEORY
c
models, relationships
BASIC NATURE THEORY
common core, theoretical foundations of science

Figure 6-2: In the domain of IT management improvements are needed in (a) artefacts, (b) applied research and (c) domain-specific theory

In the following subsections, we substantiate the desire for improvements in these three layers with informal requirements.

INFORMAL REQUIREMENTS ON ARTEFACTS (A)

CURRENT SITUATION	**DESIRED SITUATION**
System administrators often are sceptic of "fully automated" solutions.	The system should give operators the opportunity to manage the system's level of automation/ autonomy. This way, a system operator can initially choose a low level of autonomy to the system, gain experience and build up trust. Later, once accustomed to the automatism and convinced of the benefits, higher levels of autonomy can be used.
New self-management capabilities (SMC) typically only come with new systems.	We must be able to add SMC to legacy systems to protect past investments. We should not only design new systems with self-management capabilities from scratch but also link management systems with legacy systems, so that the conglomerate can be regarded self-managing
Too often security, safety, privacy added later-on.	We must build SwSMC with security, safety and privacy built-in from the start.
Too many best-effort services, too few guarantees.	Automated routines must be tested much more rigorously to avoid customer non-acceptance. We also need more formal certifications of SwSMC.
"Bad" automation can be worse than non-automation as an automated process is repeated much more often in the same time, i.e. the clock cycle is several orders of magnitude shorter.	Automated solutions must be orders of magnitude more reliable than current manual processes. If they fail, they must be built to fail safely.
IT solutions become more and more critical for life and/or property, so the impact in case of failures increases.	We must compensate the increase in fault impact by a reduction of fault probability. Therefore we have to address the most common error causes with high-availability and fault-tolerance solutions. We should also address operator mistakes.
Current IT management systems have limited context-awareness.	The system must fundamentally recognize its environment, since this will have a major influence on its future behaviour. We also must have a model of the environmental interaction, and a notion of causality.
We tell managed systems and management systems configuration parameters.	We want to increase the level of abstraction in management commands, i.e. tell policies (e.g. obligation, security), utility functions, goals and constraints to the system.
There is much heterogeneity in SwSMC.	We need to define and encourage the use of standards for models, protocols and system's behaviour.

INFORMAL REQUIREMENTS ON PROCESSES (B)

Current processes to enable automation in IT management	Desired processes to enable automation in IT management
Automation scripts are often built in an ad-hoc and non-scientific way by system administrators.	Modelling, design, implementation and testing processes must be aligned to best engineering practices.
There are typically many assumptions in system design and use. There are many side-effects, errors, bugs and vulnerabilities in current artefacts.	More proofs, verifications, validations and certifications should reduce the number of assumptions and lead to better reliability and a reduction of unwanted behaviour.
In general, IT management seems to lack an engineering tradition.	We must add a sound theoretical foundation and investigate the design, testing, simulation, and use of management systems, assistance systems, and support systems.
IT solutions builders and system administrator spend too much effort for automation. Often, each person builds his own automation solution despite significant similarities of his scenario with other scenarios.	The processes of automation in IT management should become more standardized and more efficient. We also need more formal certifications of the processes to build SwSMC.
Automation can lead to many failures in a very short time. This shortens time to become aware of potential errors. Therefore, a certain risk is associated with automation.	Automated and non-automated situation should be compared without any prejudice. This includes an economically oriented analysis of potential profit and loss, as well as a risk analysis quantifying fault probabilities and impacts.
In IT operations, faults, anomalies and misperformance are often not analyzed but worked around.	Faults, anomalies and misperformance in IT systems should be scrutinized in real operating, productive environments by instrumenting systems and learning from the results. We should get a better idea of the real behaviour of these systems and the causes of faults, anomalies and misperformance. This has partly begun with protocol analysis based on real-world traces.
There is no standard classification of types today, although the IEEE has classified a number of presumed causes for software anomalies [47].	Faults have many causes and should be classified by a number of fault types. Existing classification schemes should be used and improved.

INFORMAL REQUIREMENTS ON DOMAIN-SPECIFIC THEORY (C)

Current use of domain-specific theory	Desired use of domain-specific theory
Current IT management lacks a tradition in simulation. Few researchers in IT management have ever simulated large, complex scenarios to investigate them.	Simulation of IT environments still in its infancies, while some approaches exist, e.g. in protocol simulation (e.g. ns2) and host simulation (e.g. Bochs, Xen).
If simulation is done, usually custom programs are built.	We should use standard simulation software so that we can exchange the modelled entities.
IT management lacks behavioural models for most management entities and managed entities that could serve as input to simulators.	When creating new theory and models (domain-specific theory) we should work hard to make these new achievements directly usable in simulations.
It is incomprehensible, that computer scientists plan and build supercomputing machines, but do not make much use of it for advancing their own science, exceptions exist, e.g. simulation of complex new CPU designs.	Simulating more complex models in the same time will require more compute power. We should make more use of supercomputers for simulations.

6.2 Inspiration to achieve improvements

In the last section we have shown that we can profit from improvements at several points in IT management. This section will now categorize three different ways to approach these goals.

OBSERVATION OF ARTEFACTS IN THE REAL WORLD

One way to improve the theoretic foundations of IT management is to come up with new or improved models of reality. This may, for example, refer to the behaviour of management systems and managed systems or to their architecture. As stated in the subsection above on informal requirements on domain-specific theory all new theory and models should be formatted in a way so that they can serve as input for simulators. This way, one can check whether the combination of the simulation engine and the added model/theory really show all aspects of the real world, that are relevant in the corresponding situation. One should also check whether the new theory conflicts with existing models.

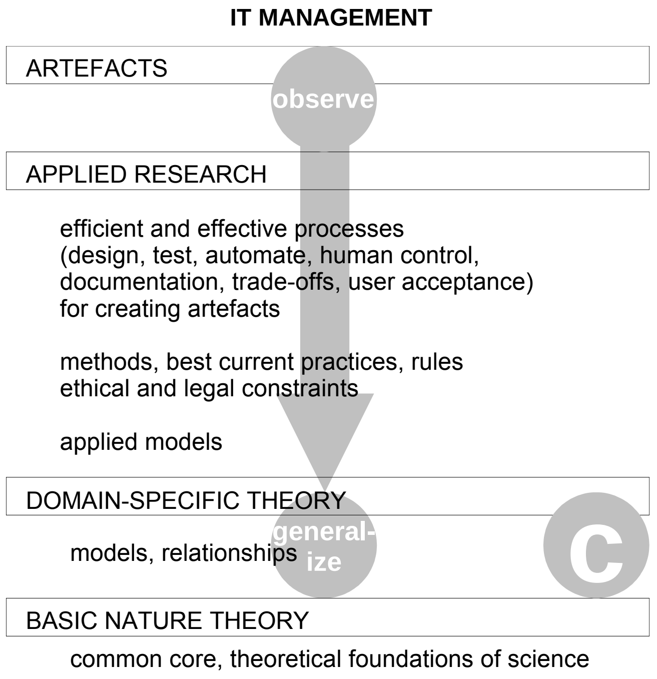

IT MANAGEMENT
ARTEFACTS
observe
APPLIED RESEARCH
efficient and effective processes
(design, test, automate, human control,
documentation, trade-offs, user acceptance)
for creating artefacts
methods, best current practices, rules
ethical and legal constraints
applied models
DOMAIN-SPECIFIC THEORY
general-
ize
models, relationships
c
BASIC NATURE THEORY
common core, theoretical foundations of science

Figure 6-3: Improvements in domain-specific theory (c) by observing and modelling architecture and behaviour of artefacts

Example:

Modelling autonomic behaviour with promise theory. Promise theory is a model of advertised behaviour that was originally introduced as a way of evaluating `cfengine` (configuration management software that adds self-management capabilities to existing computers), as current modelling techniques do not model distributed autonomous operation properly. It deals explicitly with the advertisement of decisions that have been made.

We consider a number of agents, each with private knowledge. The agents' knowledge is private and has no assumed structure, it does not necessarily have a classification according to any data model, but we assume that there exists a taxonomy of promise types that agents are assumed to agree on. Each agent has its own world view and only has access to information that promised.

A promise model is a set of promises that will lead to interactions between the agents. The behaviour of all agents might or might not be predictable from the promises made. An important question in promise theory is: can we predict how a collection of agents will behave? Unlike algorithmic approaches to modelling, such as the many that are subsumed into UML, there are no sequential mechanisms in promise theory. It is, however, possible to promise ordered activities by introducing dependencies and conditionals. If one promise is conditional on another being fulfilled, then the actions which fulfil the promises must be ordered. Promise graphs can be used to reason, like ontologies and description logics, in the sense that by following the chains of dependencies, one sees the functional processes that relate agents. One cannot take a specific fact that is not explicitly modelled however. Promises are often high-level things with the details of their implementation kept hidden. e.g. in the promise figure, we see that a history is made of observations, but that the history is also an observable.

The "promise-role" use-data by an agent could be associated with the concept of an "observer", i.e. something that samples data. This is a primitive form of reasoning. The key to the promise viewpoint is that it is based on completely autonomous behaviour. Unlike all other architectural modelling frameworks it makes no assumptions about infrastructure or cooperation. We regard this a good starting point for building models.

Promise theory is applicable to more scenarios within the IT management of IT management. Two examples include:

1) modelling peering between network service providers and
2) modelling management processes with their actors, such as IT service management processes defined in process frameworks like ITIL or eTOM.

See *http://research.iu.hio.no/promises.php* for references on Promise Theory.

Modelling Maintenance Actions to achieve Operational Predictability

Autonomics is about computer behaviour. Very little work has been done on modelling computer behaviour in computer science however. The basic hypothesis (usually not stated) is that the programmed state of a computer and its memory determines the behaviour of a computer. This can only be partially true however. The environment of the computer is not programmed and has a significant impact on its behaviour. Indeed all input to the computer is non-deterministic.

When changes occur we must determine whether they are in agreement with policy or not. This distinguishes technology from nature (natural mechanisms have no purpose, they simply operate as long as they fit in to the local ecology). Self-maintenance is a form of communication, repeating a mantra which we can call its policy, over and over again. We would like systems to chant a message that agrees with policy for its ideal state. By repeating the message one then reinforces it. This metaphor describes the idea of autonomous configuration management.

One can think of the state of the computer as a string of configuration-operational characteristics that forms an alphabet. This can be coded in any imaginable way e.g. suppose it is "ABHEKSYGHETFDH..." where A means something like `chmod 644 /etc/passwd`, etc.

After a while, corruption of the state message due to run-time interactions, users and network connections (i.e. noise) could lead to this state-message being garbled, changing some of the symbols into others. Such a change must be corrected by reiterating the actual policy. e.g. "ABCD" -> "ABXD" -> "ABCD". Just like the message over a noisy channel, we have to correct these errors. This is what an immune system does on protein strings.

Convergent operations introduced in connection with `cfengine` deal with this problem nicely. In operational language, a single operator is a unit of one kind of instigator of change, which we write

O q = q'

to mean an operation applied to a state q leads to a transition to a new state q'. But rather than thinking about a transition from one state to a new state, think of this rather as error correction. A ""convergent"" operator is a message that tells any state to transform into a policy compliant state.

C (Any state) --> (Policy-state)

The policy state is said to be a fixed-point of the policy operator since once you get there, you stay there. So, in terms of this language, all we need to do it to repeat the entire policy over and over again, like a never-ending mantra, with a separate operator for each independent kind of change.

C_1C_2C_3 ... C_n (Any state) --> (Policy-state)

The advantage of expressing maintenance and policy as convergent operations is that these can simply be repeated by a completely autonomous entity, without significant knowledge or observation about the system. The assumption is that the ideal state of the system is known in advance, however. Too little work has been done on this to answer this question today.

INSPIRATION FROM EXISTING ARTEFACTS, PROCESSES AND THEORY IN IT MANAGEMENT

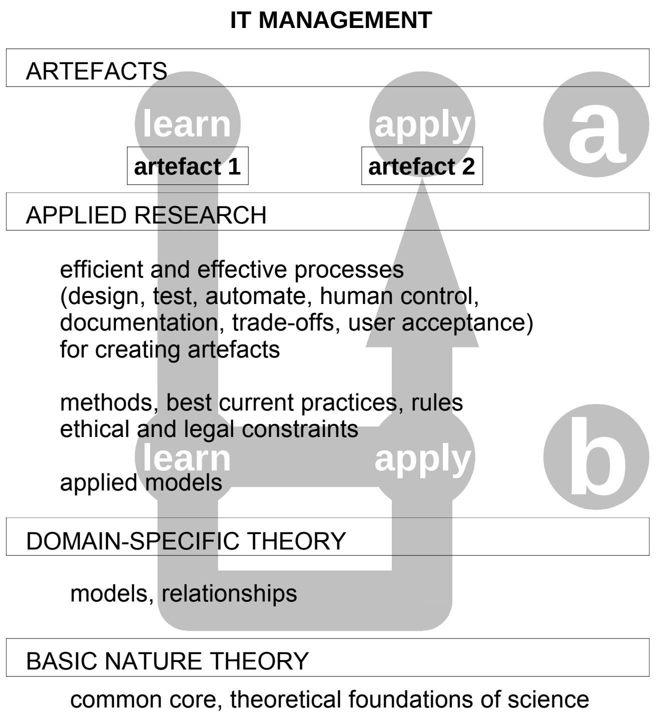

IT MANAGEMENT
ARTEFACTS
learn
apply
a
artefact 1
artefact 2
APPLIED RESEARCH
efficient and effective processes (design, test, automate, human control, documentation, trade-offs, user acceptance) for creating artefacts
methods, best current practices, rules ethical and legal constraints
learn
apply
b
applied models
DOMAIN-SPECIFIC THEORY
models, relationships
BASIC NATURE THEORY
common core, theoretical foundations of science

Figure 6-4: Single-domain inspiration from existing artefacts in IT management

This way of inspiration seems to be one of the most straightforward ones. Researchers or developers of new SwSMC in IT management learn from other existing artefacts in the same domain. However this inspiration scheme requires pioneering work to get inspired by. Clearly, patents play a major role in this context as they are designed to make new knowledge available to others while being able to earn money from past inventions for a certain period of time.

Examples: One of the first uses of self-updates in a major software product was the use of Windows Update starting with Windows 98 and then becoming more and more mature and autonomic. With Windows XP SP2 automated updates of Windows reached a state, where the self-update mechanism was enabled by default and did only marginally interfere with the end-user like asking for confirmation.

Today, many software products ranging from drivers to application software make use of automated update routines. The level of autonomy reaches from simple “new version available” notifications to automated update routines that do not interact with the user anymore.

In general, benchmarking artefacts (comparison of artefacts according to a common, generally accepted metrics) fits well in this category, if the benchmark results of the better artefact are analysed for its major causes. Blending non-conflicting features of different artefacts may result in much better artefacts.

However, we are not aware of any benchmarks for the common aspects of SwSMC in IT management.

In addition, we have identified the following artefacts of IT management as promising sources and targets of inspiration:

- management systems for and self-management in
    - mainframes
    - data center equipment such as servers and storage systems
    - network appliances such as firewalls, mail appliances, load balancers etc.
    - network devices such as routers and switches
    - operating systems and operations support systems
    - telco provider equipment
- component and service management
- patch and update management
- software distribution systems (package management systems)
- intrusion detection/prevention/reaction systems
- workflow, trouble-ticket systems
- peripherals like copiers, printers (increased usability of user interfaces guiding users to sort out failures in the device, ordering supplies)

CROSS-DOMAIN INSPIRATION FROM EXISTING ARTEFACTS, PROCESSES AND THEORY

This way of inspiration requires researchers to be aware of multiple domains of science which so far seemed unrelated to each other. They must recognize that for some problems in IT management existing knowledge from other domains can be transferred and adapted.

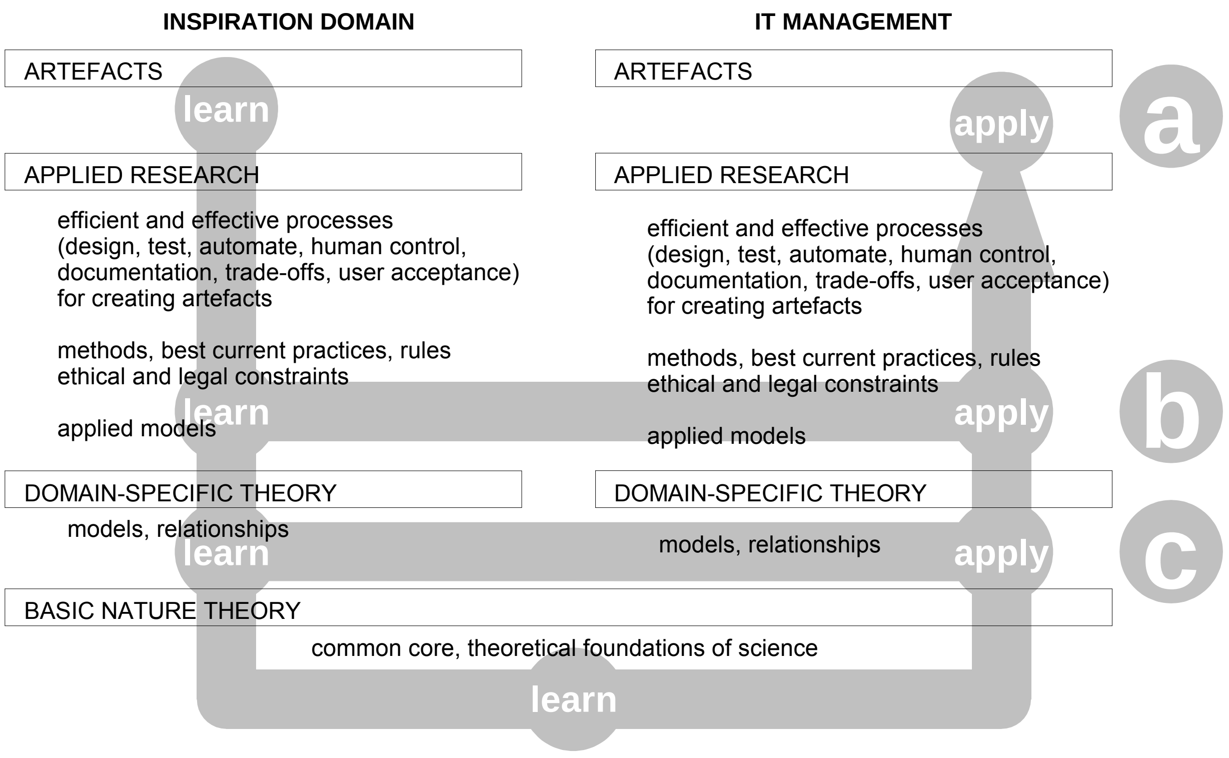

INSPIRATION DOMAIN
IT MANAGEMENT
ARTEFACTS
ARTEFACTS
learn
apply
a
APPLIED RESEARCH
APPLIED RESEARCH
efficient and effective processes (design, test, automate, human control, documentation, trade-offs, user acceptance) for creating artefacts
efficient and effective processes (design, test, automate, human control, documentation, trade-offs, user acceptance) for creating artefacts
methods, best current practices, rules ethical and legal constraints
methods, best current practices, rules ethical and legal constraints
learn
apply
b
applied models
applied models
DOMAIN-SPECIFIC THEORY
DOMAIN-SPECIFIC THEORY
models, relationships
models, relationships
learn
apply
c
BASIC NATURE THEORY
common core, theoretical foundations of science
learn

Figure 6-5: Relations between Inspiration domain and IT management

Examples: The early inspiration for autonomics came from work in **artificial immune systems** [48,49]. Applying ideas from **immunology** to configuration management, a more fully autonomic vision was later proposed independently in relation to the `cfengine` self-management system in 1998 [50]. Immune systems, a complex internal network of cooperation involving nearly every tissue in the body, are nature's approach to self-regulation (or homeostasis).

In later work IBM came up with "The Vision of Autonomic Computing” [2] being inspired by the **autonomic nervous system** of human beings. The authors state the essence of autonomic computing is system self-management, freeing administrators of low-level task management whilst delivering an optimized system. They compare their ideas to the many functions in a human body such as breathing and heartbeat that happen without humans having to be constantly aware of them. They started an “Autonomic Computing Initiative” that has since led to a good visibility of its ideas at many conferences.

Many basic algorithms in self-organization are inspired by schools of fish or ant colonies. As we can see, **biology** was used a source of inspiration many times.

Concepts like machine learning, reasoning and inference, planning and the Monitor-Analyze-Plan-Execute loop based on Knowledge as defined by IBM in the Autonomic Computing Initiative are borrowed from the domain of **artificial intelligence**.

The use of policies was inspired by **politics, law and business administration.**

Last but not least, **sociology** contributes with knowledge in group behaviour and organizational structures.

6.2.1 Inspiration from engineering

As said in the previous section, "artefacts of nature" such as the human body or animals, while not created by humans, have often been the source of inspiration for technical SwSMC. However, even if techniques like genetic and evolutionary algorithms are applied as optimization techniques, we cannot copy nature's way to create its objects.

Therefore, we should regard the tuple (Current Deficiencies, Goals) from Table 1 as a complex pattern and look for pattern matches in the development of other domains. We can assume that other domains have gone through similar steps towards automation and operator support, where system operators / users have been released from routine tasks by management systems, assistance systems, support systems. Therefore, we should add **engineering** domains to our inspiration domains.

INSPIRATION FROM ENGINEERING DOMAINS

We came up with the following list of engineering domains and promising artefacts, which seem to fulfil the requirements of successfully applied automation to achieve self-management capabilities:

Supply networks

- management systems for (nuclear) power plants or related facilities (Reliability and fault analysis)

Medical devices and appliances

- life support machines, surgery support systems, telemedicine, medical expert systems

Military

- "smart" weapons, unmanned systems, combat aerial or ground vehicles
- combat management and support systems
- reconnaissance systems

Security and surveillance

- surveillance, monitoring systems, alarm and response systems
- early warning systems (earth quake, seaquake, tsunami, ABC hazards, fire)
- sensor networks

Logistics and Transportation

- telemetric services

Robotics Engineering

- robots, multi-robot systems, multi-agent systems

Automotive and Railway Engineering

- assistant systems such as ABS, ESP, assistant systems enabled by *-by-wire techniques, engine control, parking assistants
- train assistant systems
- remotely operated trains and cars
- communication network connectivity

Space and Aviation Engineering

- unmanned space objects, such as satellites
- communication network connectivity
- autopilot, fly-by-wire systems, ground control, tower instrument landing system
- flight control systems
- remotely operated aircraft

Ships Engineering

- zero-maintenance (reduced maintenance) efforts
- unmanned underwater vehicles
- remotely operated vehicles (ROV)

Production plants, Industrial automation

- (IT-assisted) automation in manufacturing systems

IDENTIFICATION OF AUTOMATION CONCEPTS IN ENGINEERING

In the way described above, we want to:

- look for existing systems (artefacts) and system design (processes) from "inspiration domains" that make plausible the approach of goals mentioned in Table 6-1
- examine their foundations, design, search for concepts, solutions, best practices,
- apply the found concepts to the target domain of IT management.

This way we can leverage their existing knowledge and experience and apply successful concepts to IT management achieving improvements in (c) theory, (b) processes and (a) artefacts.

To identify the concepts that we are looking for we compiled the following list of questions, that engineering domains must have come across and must have come up with solutions or design principles:

Why to automate?

- How to find the most suitable scenarios to apply automation?
- What are the reasons for the application of management automation?
- How to improve automation benefit and reduce automation risk?
- What legal and ethical constraints pose limits on systems designs?

How to automate?

- Where to apply remote management, where self-management?
- Are there classification approaches for autonomy levels?
- How should SwSMC cooperate (autonomy vs. cooperation)?
- How to design efficient and effective processes (design, test, automate, documentation, trace-offs) in SwSMC?
- What kind of control should human and non-human managers be able to exercise on SwSMC?
- How should humans and machines cooperate?
- How to cope with complex models?

What are **customer requirements** on the design of SwSMC?

- How to calculate or estimate operational expenditures of SwSMC in a generally accepted way?
- How to build up, improve and then keep customer acceptance of SwSMC?

6.3 Outlook

Future work will cover aspects of autonomic systems that are independent of the actual application domain. Our aim would be to a) investigate the potentials and risks of systems with self-management capabilities, b) introduce a cross-domain evaluation scheme to compare systems with self-management capabilities and c) give an overview on enabling technologies to implement autonomic management components.

7 Conclusion

Having investigated in depth Next Generation Management technologies and approaches to support Autonomic Management, we conclude that the first steps towards Autonomic Management have been made. The outcome of ongoing research integration and collaboration among partners of EMANICS Network of Excellence and in particular Work Package 9 (WP9: Autonomic Management) as presented in this document provides an overview of current trends and available technologies that can realise the vision of autonomic computing.

Policy based techniques and in particular policy refinement is intimately linked to the principles of Autonomic Management. Policy-based management can effectively address the issues of Self-configuration and Self-optimization as two of the key properties of Autonomic Systems. These are illustrated in realistic case studies presented in subsequent sections that demonstrate the applicability of the policy refinement techniques described in this chapter in a realistic application for Quality of Service provisioning and management in IP Networks. We have described the use of abductive reasoning and finite state transducers concepts for formalising both authorisation and management policies and performing analysis of policy-based systems. Although our focus has been primarily on conflict detection, checking that a given policy specification satisfies well-defined properties follows an identical pattern to the check for application specific conflicts. Although the underlying formal representation and reasoning are not easily accessible to users that are not well versed in logic programming, the tools we have implemented hide the underlying complexity and automatically derive the formal representation from the policies themselves and from design-level models of managed objects' behaviour such as statecharts.

The significance of Web Services for Management has been explained and they have been identified as an appealing enabler of Autonomic Management. Some of the unique features of Web Services can assist in achieving self-configuration, self-awareness and self-optimisation, like increased performance, platform independence and interoperability, easy deployment on heterogeneous networks as well as a wealth of tools and development potentials. Obviously there are great improvement margins for Web Services based management. Standardisation efforts are continuing and additions to the framework are being developed. It is evident though that there is significant potential in the adoption of Web Services for Autonomic Management.

Trust and misbehaviour are major issues for designing and developing the models and architectures of efficient autonomic systems. We detailed the underlying key concepts and showed how they can be applied to the case study of ad-hoc networks. First, the autonomic systems are required to describe themselves to determine their behavioural properties. These descriptions can be both internal when an autonomic system specifies its own properties using promises, and external when an autonomic system specifies the properties it expects from another system. Then, the systems monitor themselves to determine if their behaviours are normal or anomalous. The goal is to detect faults at a short term and estimate reliability of systems at a longer term. Two analysis methods were proposed to synthesize the data set of observations measured by the systems. The *k*-means analysis is capable of clustering this data set to separate the reliable systems from the unreliable ones. The principal components analysis is capable of simplifying the data set by retaining the most significant characteristics. Based on these analyses, the final decision to trust other autonomic systems is essentially a policy decision. The valuation of a system's trustworthiness can then be propagated among the other systems in the form of reputation. However, in both cases trust and reputation, the semantics of the final value are a matter of the policy of the final recipient. Finally, we presented how misbehaviour can influence trust and reputation measurements in autonomic systems/components and examined a case study based on the principle of conservation of flow.

We have also presented recent efforts in exploiting context awareness towards Autonomic Management. The CoCo Infrastructure, like many other architectures for context-aware systems, follows an infrastructure-centred distributed services model based on client-server dialog. It acts as a broker between context-aware services (CAS) that request context information and context information services (CIS) that provide context information. It therefore relieves the context-aware services from the burden of discovering context information services, data transformation, or derivation of high-level context information from low-level context information. In addition, we have demonstrated how context information can be combined with the concepts of patterns and become an important part of distributed computing and processing with cfengine.

Finally, we have explained how we can expedite progress in systems with self-management capabilities (management systems, assistance systems, support systems) in IT management by getting inspiration from existing knowledge in IT management itself or other domains. After introducing a model for the knowledge in and across domains, we have classified ways of inspiration and listed promising inspiration domains. These ideas provide further motivation for our future work. We intend to cover aspects of autonomic systems that are independent of the actual application domain. Our aim would be to a) investigate the potentials and risks of systems with self-management capabilities, b) introduce a cross-domain evaluation scheme to compare systems with self-management capabilities and c) give an overview on enabling technologies to implement autonomic management components.

Having explained how policy-based approaches to network and systems management are of particular importance. They provide autonomic behaviour in the form of an efficient closed feed-back loop either locally or hierarchically. However, policy based techniques are not in widespread use because their advantages in terms of short term return on investment are difficult to justify without significant advances on tools for policy analysis and refinement. In our future work, we will investigate in depth the issue of policy refinement. Policy refinement is the process of transforming a high-level, abstract policy specification into a low-level, concrete one. The main objectives of a policy refinement process are (1) determine the resources that are needed to satisfy the requirements of the policy, (2) translate high-level policies into operational policies that the system can enforce and (3) verify that the lower level policies actually meet the requirements specified by the high-level policy. Once these objectives are met, we will be able to provide a powerful tool for policy-based management and will establish the applicability of PBM for supporting Autonomic Management.

8 References

[1] Haas, R., Droz, P. and Stiller, B., “Autonomic service deployment in networks”, IBM Systems Journal, Vol. 42, No 1, 2003

[2] Kephart, J.O. and Chess, D.M., “The Vision of Autonomic Computing”, IEEE Computer, January 2003

[3] Ganek, A. G. and Corbi, T.A., “The dawning of the autonomic computing era”, IBM Systems Journal, Vol. 42, No 1, 2003

[4] N. Damianou, N. Dulay, E. Lupu and M. Sloman. The Ponder Policy Specification Language. Policy Workshop 2001, Jan. 2001, Bristol, U.K., Springer-Verlag, LNCS 1995

[5] Sloman, M. (1994). Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, 2(4):333-360, Plenum Press.

[6] R. A. Kowalski and M. J. Sergot, "A logic-based calculus of events," New Generation Computing, vol. 4, pp. 67-95, 1986.

[7] R. Miller and M. Shanahan, The Event Calculus in Classical Logic Alternative Axiomatisations, vol. 4. Linkoping, Sweden: Linkoping University Electronic Press - http://www.ep.liu.se/ea/cis/1999/016/, 1999.

[8] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer, "An Abductive Approach for Analysing Event-Based Requirements Specifications," presented at 18th Int. Conf. on Logic Programming (ICLP), Copenhagen, Denmark, 2002

[9] B. Beckert, U. Keller, and P. H. Schmitt, "Translating the Object Constraint Language into First-order Predicate Logic," presented at VERIFY, Workshop at Federated Logic Conferences (FLoC), Copenhagen, Denmark, 2002.

[10] E. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management, 25(6):852-869, Nov./Dec. 1999

[11] J. D. Moffett and M. S. Sloman, "Policy Conflict Analysis in Distributed System Management," Journal of Organisational Computing, vol. 4, pp. 1-22, 1994.

[12] D. F. C. Brewer and M. J. Nash, "The Chinese Wall Security Policy," presented at IEEE Symposium on Research in Security and Privacy, Oakland, California, USA, 1989.

[13] J. Baliosian and J. Serrat , “Finite State Transducers for Policy Evaluation and Conflict Resolution”, Proceedings of the IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004) Yorktown Heights, New York, 7-9 June 2004

[14] G. van Noord and D. Gerdemann, “Finite state transducers with predicates and identities,” *Grammars*, vol. 4, pp. 263–286, December 2001

[15] K. McCloghrie and F. Kastenholz, “RFC 2863: The Interfaces Group MIB”, IETF, June 2000.

[16] J. Schönwälder, A. Pras and J.P. Martin-Flatin, “On the Future of Internet Management Technologies”, *IEEE Communications Magazine*, Vol. 41, No. 10, pp. 90-97, Oct. 2003

[17] J. v. Sloten, A. Pras, M.J. v. Sinderen: “On the standardisation of Web service management operations”, EUNICE 2004 - Proceedings of the 10th Open European Summer School and IFIP WG6.3 Workshop (Eds: Harjo J., Moltchanov D. and Silverajan B.), ISBN: 952-15-1187-7, Tampere, Finland, June 2004, page 143-150

[18] M. Choi, J.W. Hong, H. Ju: “XML-Based Network Management for IP Networks”, ETRI, Journal, Volume 25, Number 6, December 2003, pp. 445-463

[19] M. Choi, J.W. Hong: “Performance Evaluation of XML-based Network Management”, Presentation at the 16th IRTF-NMRG meeting, Seoul, Korea, 2004, http://www.ibr.cs.tu-bs.de/projects/nmrg/meetings/2004/seoul/choi.pdf

[20] R. Neisse, R. Lemos Vianna, L. Zambenedetti Granville, M. Janilce Bosquiroli,Almeida, L. Margarida Rockenbach Tarouco: “Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways”, Proceedings of NOMS2004, Seoul, Korea, 14 pages

[21] T. Drevers, R. v.d. Meent and A. Pras, “Prototyping Web Services based Network Monitoring”, in J. Harjo, D. Moltchanov and B. Silverajan (Eds.), Proc. 10th Open European Summer School and IFIP WG6.3 Workshop (EUNICE 2004), Tampere, Finland, June 2004, pp. 135–142

[22] G. Pavlou, P. Flegkas, S. Gouveris: “Performance Evaluation of Web Services as Management Technology”, Presentation at the 15th IRTF-NMRG meeting, January 8 2004, Bremen, Germany

[23] G. Pavlou, P. Flegkas, S. Gouveris, A. Liotta: “On Management Technologies and the Potential of Web Services”, IEEE Communications, Vol. 42, No. 7, July 2004

[24] A. Pras, T. Drevers, R. v.d. Meent and D. Quartel, “Comparing the Performance of SNMP and Web Services-Based Management”, IEEE e-Transactions on Network and Service Management (eTNSM), Vol. 1, No. 2, December 2004

[25] R. v. Engelen, “gSOAP Web services toolkit”, 2003, http://www.cs.fsu.edu/~engelen/soap.html

[26] R. v. Engelen, K. A. Gallivany: “The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks”, Proceedings of IEEE Cluster Computing and the GRID 2002, , Berlin, Germany, 2002, pp. 128-135

[27] Zlib homepage: http://www.zlib.org

[28] Local and Global Trust Based on the Concept of Promises J. Bergstra and M. Burgess, (submitted) 2006

[29] Scaling behaviour of peer configuration in logically ad hoc networks M. Burgess and G. Canright. IEEE eTransactions on Network and Service Management, p1 (2004)

[30] An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation M. Burgess. Lecture Notes on Computer Science, 3775:97-108, 2005

[31] Fault Monitoring Based on Information Theory R. Badonnel, R. State and O. Festor. In Proc. of the 5th International IFIP-TC6 Networking Conference (NETWORKING'06), p. 427-438, Coimbra, Portugal, 2006.

[32] Self-Configurable Fault Monitoring in Ad-Hoc Networks R. Badonnel, R. State and O. Festor. Elsevier Journal of Ad-Hoc Networks, to be published in 2007.

[33] Optimized Link State Routing Protocol (OLSR) T. Clausen, P. Jacquet. IETF Request For Comments 3626, 2003

[34] An Algorithm to Detect Packet Forwarding Misbehavior in Mobile Ad-Hoc Networks O. F. Gonzalez, M. Howarth and G. Pavlou. In Proc. of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM2007), Munich, May 2007.

[35] Detection of Packet Forwarding Misbehavior in Mobile Ad-Hoc Networks O. F. Gonzalez, M. Howarth and G. Pavlou. In Proc. of the 5th International Conference on Wired/Wireless Internet Communications, Coimbra, May 2007.

[36] Some Methods for Classification and Analysis of Multivariate Observations J. B. MacQueen. In Proc. of the 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, CA, 1967, pp. 281–297

[37] K-means Clustering via Principal Component Analysis C. Ding and X. He. In Proc. of International Conference on Machine Learning (ICML 2004), pp 225-232. July 2004.

[38] Detecting disruptive routers: a distributed network monitoring approach K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. In Proc. of the 1998 Symposium on Security and Privacy, pp. 115-124, May 1998.

[39] SCAN: Self-organized network-layer security in mobile ad hoc networks H. Yang, J. Shu, X. Meng, and S. Lu. IEEE Journal on Selected Areas in Communications, vol. 24, issue 2, pp. 261-273, February 2006.

[40] Providing robust and ubiquitous security support for mobile ad-hoc networks J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Proc. of the 9th IEEE International Conference on Network Protocols, pp. 251-260, November 2001.

[41] Securing ad hoc networks L. Zhou, and Z. Haas. IEEE Network Magazine, vol. 13, issue 6, November/December 1999.

[42] EMANICS consortium: Deliverable 9.1 – Frameworks and Approaches for Autonomic Management of Fixed QoS-enabled and Ad Hoc Networks

[43] Winograd, T. (2001): Architectures for Context. In Human-Computer Interaction, 16 (2) p. 401-419.

[44] Buchholz, T., Krause, M., Linnhoff-Popien, C., Schiffers, M.: CoCo: Dynamic Composition of Context Information. Proceedings of the First Annual International Conference on Mobile and Ubiquitous Computing (MobiQuitous), August 2004.

[45] Fuchs, F., Hochstatter, I., Krause, M., Berger, M.: A Meta–Model Approach to Context Information. Proceedings of 2nd IEEE PerCom Workshop on Context Modeling and Reasoning (CoMoRea) (at 3rd IEEE International Conference on Pervasive Computing and Communication (PerCom 2005)), March 2005

[46] Hochstatter, I., Duergner, M., Krause, M.: A Context Middleware Using an Ontology-based Information Model. Proceedings of the EUNICE Summer School, July 2007.

[47] A Standard Classification for Software Anomalies, IEEE Press, 1992

[48] Self-Nonself Discrimination in a Computer. S. Forrest, A.S. Perelson, L. Allen, R. and Cherukuri. In Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA: IEEE Computer Society Press (1994)

[49] A Sense of Self for Unix Processes. S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. In Proceedings of 1996 IEEE Symposium on Computer Security and Privacy (1996)

[50] Computer Immunology, M. Burgess, Proceedings of the Twelfth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA), page 283, (1998)

9 Abbreviations

AODV	Ad Hoc on Demand Distance Vector
CAS	Context Aware Service
CLI	Command Line Interface
EC	Event Calculus
FST	Finite State Transducers
GAP	Generic Aggregation Protocol
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
LDAP	Lightweight Directory Access Protocol
MANET	Mobile Ad Hoc Network
OLSR	Optimized Link State Routing
PBNM	Policy-Based Network Management
PCF	Principle of Conservation of Flow
PDA	Personal Digital Assistant
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PMT	Policy Management Tool
PR	Policy Repository
QoS	Quality of Service
RPC	Remote Procedure Call
SLA	Service Level Agreement
SLS	Service Level Specification
SNMP	Simple Network Management Protocol
SwSMC	Systems with Self-Management Capabilities
TF	Tautness Functions
TFFST	Finite State Transducer with Tautness Functions and Identities
UCS	Ubiquitous Computing System
UML	Unified Modelling Language
WLAN	Wireless Local Area Network
WS	Web Services
WSDL	Web Service Definition Language
XML	eXtensible Markup Language

10 Acknowledgements

This deliverable was made possible due to the large and open help of the WP9 team of the EMANICS consortium within the Network of Excellence, which includes all of the deliverable authors as indicated in the document control. Many thanks are owed to all.