J Netw Syst Manage (2009) 17:255–284
DOI 10.1007/s10922-009-9126-4

The Use of Context-Aware Policies and Ontologies to Facilitate Business-Aware Network Management

John Strassner · Sven van der Meer · Declan O'Sullivan · Simon Dobson

Published online: 19 May 2009
© Springer Science+Business Media, LLC 2009

Abstract The purpose of autonomic networking is to manage the business and technical complexity of networked components and systems. However, existing network management data has no link to business concepts. This makes it very difficult to ensure that services offered by the network are meeting business objectives. This paper describes a novel context-aware policy model that uses a combination of modeled and ontological data to determine the current context, which policies are applicable to that context, and what services and resources should be offered to which users and applications.

Keywords Context · Ontology-based management · Policy management · Semantic reasoning

1 Introduction

The business, technical, and even social aspects of systems have increased dramatically in complexity, requiring new technologies, paradigms and

J. Strassner (✉) · S. van der Meer
TSSG, Waterford Institute of Technology, Carriganore Campus, Carriganore, County Waterford, Ireland
e-mail: jstrassner@tssg.org

S. van der Meer
e-mail: vdmeer@tssg.org

D. O'Sullivan
Trinity College Dublin, Dublin, Ireland
e-mail: declan.osullivan@tcd.ie

S. Dobson
University College Dublin, Dublin, Ireland
e-mail: simon.dobson@ucd.ie

functionality to be introduced to cope with these challenges [1]. This increase in complexity has made it almost impossible for a human to manage the different operational scenarios that are possible in today's communication systems. While IP network management problems have been extensively documented [1–4], wireless systems present even more difficult problems [5]. For example, wireless failures are usually not obtainable from a single set of attributes—they must be *inferred*. Cognitive networks [6], in which the type of network access can be dynamically defined, face additional constraints, such as regulatory compliance.

One of the underlying problems in network management is the lack of a *lingua franca* to ensure that vendor devices having different functionality and data models can be programmed to provide the same resource or service.[1] Part of the allure of Policy-Based Network Management (PBNM) [7] is its simplicity in providing different services to different users while automating device, network and service management in a standard way. However, most PBNM systems have been low-level systems that manage changes in commands for routers, switches, and firewalls. Hence, there is no link between business needs and the configuration of network resources and services.

In addition, while context has been examined as a method to determine changes in functionality [8, 9], it has not yet been related to business changes. There are two reasons for this. First, the act of monitoring current context and operation is difficult to perform in distributed systems [10]. Second, different sensors have different reference models and management data; these data must be *fused* in order to develop an accurate and informed model of context [11, 12].

However, even if the above are solved, an important problem remains: how does the business benefit from this? Business objectives must be able to determine which network services and resources should be made available to a given set of users and/or applications at a particular time.

This paper describes a novel context-aware policy model that uses a combination of modeled and ontological data to determine the current context, which in turn determines the set of policies that are applicable to that context. These policies are then used to determine the set of roles [13, 14] that are *permissible* to use, which in turn defines the specific set of services and resources that should be offered.

The FOCALE autonomic architecture [15] was built in part to solve this problem, and is based on four key concepts. First, the use of a shared information model is required in order to harmonize the different data models that are used in Operational and Business Support Systems (OSSs and BSSs). Second, information and data models are augmented with ontological data, which enables the representation and use of semantics to reason about behavior. This novel combination of modeled and ontological data enables reasoning to be used to define the best set of policies applicable for a given context change. Third, we define a context-aware policy model, which enables context changes to trigger the use of new policies that can adapt offered resources and services to sensed context changes. Finally, we outline

[1] See, for example, the recent IETF NetMod working groups discussions on conformance. For example, P. Shafer wrote on 4/10/08: "Or, viewed the other way, the assumption that vendors will faithfully implement data models as specified has been harmful to the adoption of network management."

how this novel context-aware policy architecture can be used, together with machine learning and reasoning, to link business objectives to the use of network services and resources.

The organization of this paper is as follows. Sections 2 and 3 summarize current PBNM and context approaches. Section 4 describes the Policy and Knowledge Continua, which are two abstractions that link business concepts to network services and resources. Section 5 describes our context-aware policy model. Section 6 provides an example of mapping to business concerns, and Sect. 7 examines the problems of stability and uncertainty. Section 8 summarizes the paper and future work.

2 Current PBNM Approaches

Policy-based network management (PBNM) is a concept developed originally to reduce the administrative complexity of reconfiguring a device and/or a network to respond to the changing conditions of the business and the infrastructure. The manual process of reconfiguring the network is very difficult; two important sources of this difficulty are the challenge of enabling the business to determine the set of network services and resources to be provided at any given time, and because of the vast amount of heterogeneous devices a network comprises [5]. PBNM aims to decrease this complexity and its associated cost by automating, to some degree, the reconfiguration process. This concept corresponds to the vision of self-governance innate in autonomic communications [16].

Damianou et al. [17] provides a comprehensive survey of policy management approaches. One of the key points that are raised in [17] is that the capabilities and content of a given policy language is almost always bound to a specific domain. For example, the syntax, constructs used, and underlying model are very different between existing security and management policy languages. Three languages that claim to be independent of domain are briefly discussed below. Note the following shortcomings of all of these related works:

None address the multiple stakeholders and views embodied in the Policy Continuum concept
None provide a common *lingua franca* that enables different management data to be harmonized
All are limited in their representation of knowledge compared to our method, which combines information models and ontologies
None relate business objectives to network services or resources.

The policy technologies group at the IBM T.J. Watson Research Centre developed the Policy Management for Autonomic Computing (PMAC) technology, which gave rise to the Apache Imperius project.[2] This approach only uses a condition–action tuple, which creates efficiency and predictability problems. The

[2] Imperius was accessed from the following URL on October 4, 2008: http://incubator.apache.org/imperius/.

most obvious is that if policy rules do not contain an event specification, then the system cannot determine when conditions will be evaluated. This in turn means that conflicts cannot be detected. In addition, continuously checking every condition against every event is not going to be computationally efficient, nor will it produce temporally predictable firing of policy actions without prescriptive real-time requirements. Finally, the predictability problems that may be introduced can open the door to a wide range of anomalous runtime behaviors.

Ponder [18], a policy language for distributed system management, has been developed as part of ongoing work in Imperial College, London. The language, which is declarative, strongly typed and object-oriented, codifies six policy types, namely positive authorization, negative authorization, refrain, positive delegation, negative delegation and obligation. While these policy types allow for a rich policy set to manage the behavior of a domain, the language has some shortcomings. Both positive and negative authorizations are target based, which means that a subject cannot specify positive authorization policies to control its own behavior regarding interaction with a target. Refrain policies are approximately equal to subject based negative authorization policies, although the semantics of 'must refrain from' are not as strong as 'not authorized'. Moreover, refrain, positive authorization and negative authorization policies do not have an event clause, which can cause efficiency and predictability problems as discussed in relation to PMAC. Delegation policies, both positive and negative, allow the subject of an authorization policy to delegate temporarily access rights to a grantee. However, as authorization policies are target based, it therefore seems imprudent to allow the subject of an authorization policy to delegate permission/revocation access rights. Finally, the semantics of the Ponder policy language itself have not been formally defined, making automated reasoning over policies, as required for autonomic networking, not possible through formal means.

The University of Maryland, Baltimore, has developed Rei [19] to facilitate the definition of deontic logic based policies that are used to manage the behavior of agents operating in open distributed environments. More specifically, Rei defines constructs for right, prohibition, obligation and dispensation policies. However, unlike Ponder, Rei does not specify explicitly if a policy is target based or subject based, which can be limiting for policy conflict detection. OWL-Lite is used to encode the grammar of Rei; therefore individual instances of policies are defined as individuals/slots of the defined classes and properties. Rei also uses OWL-Lite to reason over domain knowledge expressed in either RDF or OWL. To overcome the issue of defining policies that contain variables, which is inherited by implementing OWL to encode the grammar, Rei uses placeholders similar to those used in Prolog. This non-standardized extension, however, means that (DL Implementation Group) DIG reasoners and the REI engine are able to reason about the domain-specific knowledge, but not about all policy specifications. Finally, the most important critique of the Rei policy specification language is that the semantics of the language have not been formally defined. A natural language description (English) of the semantics has been provided but this is of little benefit for automated knowledge acquisition and inferencing.

3 Current Context Approaches

Context-aware applications are gaining increasing recognition as a means to realize the needs of current and next generation applications. The state of the art in context awareness is again limited, in that existing approaches do not take into account relating business needs to network resources and services. In particular:

None address the multiple stakeholders and views embodied in the Policy Continuum concept

None use the combination of ontological data and policy rules to jointly define functionality that can be used for that particular context

None relate business objectives to network services or resources

The development of software architecture to support the building of context-aware applications is described in [9]. While the framework simplifies the task of acquiring and delivering context to applications, it does not provide an information model to visualize or define how context is used, nor does it describe how policy is used. It also does not relate business objectives to services or resources.

Román et al. [20] defines a context service to let applications query and register for particular context information using a first-order logic to model context. This limits applications to using only context data from the context providers defined in the registry. Again, no use of policy or association with business objectives is provided.

Most context-aware approaches focus on building frameworks and toolkits to support ad hoc context representation [21]. Hence, [21] describes an ontology for describing concepts for context-aware applications. However, it is designed to support context negotiation, not general purpose analysis, and does not use policy or map business concepts to network services.

There are several proposed extensions to the Composite Capabilities/Preferences Profile (CC/PP)[3] and User Agent Profile (UAProf)[4] standards. Both of these mention context, but are in reality focused on describing device capabilities and user profiles. In addition, they do not define policy rules or how business concepts are mapped to context.

One of the most popular definitions of context is [8]: “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves”. For our purposes, this definition has some significant shortcomings. First, context involves more than “just” data or information—it is about the use of *knowledge*. Data is simply a set of values, with no underlying understanding of what those values ultimately mean. Data is transformed into information when different meanings are associated with information that is context-specific. When contextual information is interpreted and understood using one or more interpretation rules, we then have knowledge. We

[3] W3C, Composite Capabilities/Preferences Profile (CC/PP), http://www.w3.org/Mobile/CCPP.

[4] WAPFORUM, User Agent Profile (UAProf), http://www.wapforum.org.

explicitly model knowledge using a combination of models and ontologies, and update this knowledge using machine learning and reasoning.

Second, the phrase "characterize the situation of an entity" is too limiting. For example, if the context was based on the current heart rate of a patient, the situation of the patient may or may not have anything to do with determining the value of the context data. Hence, we have built an extensible model, based on a set of software patterns,[5] to represent an aggregate model of context, where the specific aspects of knowledge to be aggregated are determined by context-aware policies.

The third problem is the lack of formal definitions of concepts that involve and interact with context. For example, the phrase "An entity is a person, place or object" includes only some concepts that are relevant, and does not include concepts such as communication, walking, or being in a noisy environment. Hence, we base our context model definition in an object-oriented information model constructed from software patterns—we are thus able to leverage the work in the information model to dynamically adjust the definition of context to meet application-specific needs by modeling context as a set of reusable contexts.

The phrase "interaction between a user and an application" is problematic, since context exists independent of the interaction between a user and an application (e.g., the fact that no one is walking, sitting, or standing on a pressure-sensitive floor is context in and of itself).

Finally, when we look at the entire second sentence of this definition ("An entity…that is considered relevant to the interaction…"), this means that only people, places or objects that are relevant to the interaction between a user and an application are considered context. This is clearly wrong for autonomics, as well as ubiquitous computing and other similar approaches, since such approaches emphasize the concept of invisible management. In other words, interaction should not have to be explicit and noticed by the user.

Ye et al. [22] provides a detailed review and critique of ontologies that have been developed for pervasive systems. The paper explains the two main applications of ontologies in pervasive computing: modeling context and reasoning about it. The defining characteristics of context—its dynamism, incompleteness, openness and uncertainty—have proven to be extremely difficult to capture in orthogonal and portable form. Intuitively one would like to perform a "separation of concerns", isolating (for example) the model of uncertain reasoning from the sensors whose readings were being reasoned over. Such separations are not typical in current ontologies, since most ontologies are constructed to the "the complete answer" to a given problem domain. We have taken a first step towards solving this problem in the DEN-ng context model used in FOCALE, which is explained in Sect. 5.4.

We use the following definition of context in DEN-ng: "The Context of an Entity is a collection of measured and inferred knowledge that describe the state and environment in which an Entity exists or has existed". In particular, our definition emphasizes two types of knowledge—facts (which can be measured) and inferred data, which results from machine learning and reasoning processes applied to past and current context. Both of these concepts are realized as intelligent containers of

[5] See, for example: Fowler, analysis patterns—reusable object models, ISBN 0-201-89542-0.

information, which can have additional semantics added to them, such as metadata, accuracy, and measurement confidence. They also include context history, so that current decisions based on context may benefit from past decisions, as well as observation of how the environment has changed. Our approach uses models and ontologies to overcome these and other problems by providing a richer expression of semantics than UML is capable of.

4 The Policy and Knowledge Continua

Data is characterized as observable and possibly measurable raw values that signal something of interest. In and of themselves, data have no meaning—they are simply raw values. In contrast, data is transformed into information when meaning can be attached to data. The process of transforming information into knowledge attaches purpose, context, and provides the potential to generate action.

For example, a measured value of 17 is simply a scalar. If that value can be associated with a set of enumerated values, and if the value 17 has a particular meaning (e.g., “problem”), the system has now succeeded in attaching a meaning to the scalar value. If the system can add further details, such as what the problem refers to and what a possible solution to this problem could be, the semantics are now made more explicit, and important additional information and knowledge can now be generated (i.e., a skill set could be inferred as required to solve the problem whose value was denoted as 17). This systematic enrichment of semantics, from a simple value to a value with meaning to a value with meaning, purpose and action, is critical to defining knowledge that can be used to perform a task. The Entity, Value, and MetaData subclasses of the DEN-ng information model shown in Fig. 1 provide these semantics.

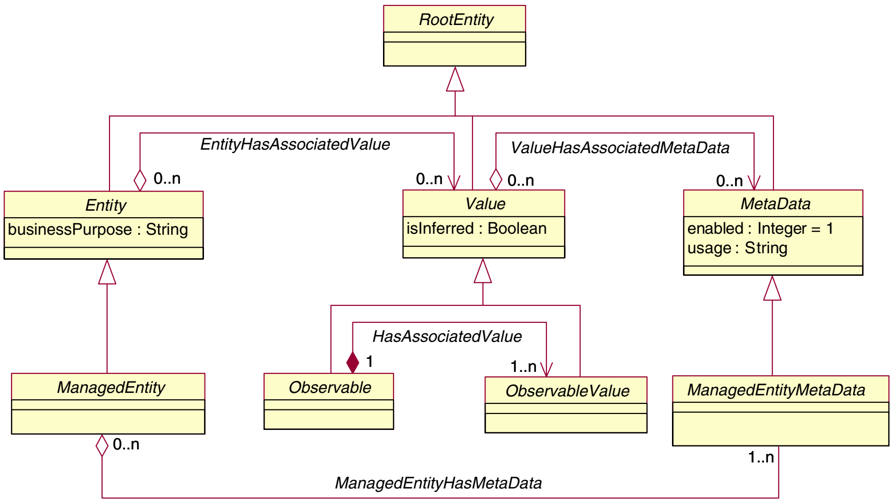


Fig. 1 Difference between Entities, Values, and Metadata in DEN-ng

The Entity hierarchy represents classes of objects that play a business function. Entities can be either managed or unmanaged, and represent objects that have a separate and distinct existence (as opposed to being merely a collection of attributes or an abstraction of behavior). In contrast, the Value hierarchy consists of subclasses that are used to reify the notion of something that exists that does not have a distinct associated identity (like subclasses from Entity do). Note the separation between Observable and ObservableValue—this decouples values from objects, such as containers, in which the values exist. This enables either or both to have metadata.

DEN-ng differentiates between something that is observable (but not necessarily measurable) and something that is both observable and measurable. Hence, the Measurable and MeasurableValue classes are subclasses of the Observable and ObservableValue subclasses (which are not shown in Fig. 1 for the sake of simplicity). Thus, all MeasureableValues are ObservableValues, but not all ObservableValues can be measured.

Metadata represents information that describes or explains the purpose, operation, characteristics, interaction, and/or behavior of the entity that the Metadata is applied to. The MetaData hierarchy exists as a third, and separate, hierarchy because it can be applied to Entity as well as Value objects. This extensible design is unique in the industry.

There is a profound difference between modeling a fact that is *observed* or *measured* and modeling a fact that is *inferred*. Facts that are observed or measured often do *not* need additional reasoning performed on them, as the value(s) that they represent are sufficient to define the fact. In contrast, facts that are *inferred* can only exist by having reasoning performed to create them. The general approach for representing inferred knowledge in the DEN-ng information model is shown in Fig. 2.

The figure shows the six attributes that make up the generic class container used in DEN-ng to store inferred knowledge. The typeOfContainer attribute is an enumerated integer, and defines which type of pre-built or application-specific container is used. This enables the developer to attach application-specific metadata to different container types. The actual data that is inferred is contained in the inferredContent attribute. It is stored as an OctetString to enable different types of results to be stored independent of any one particular data type, and works in

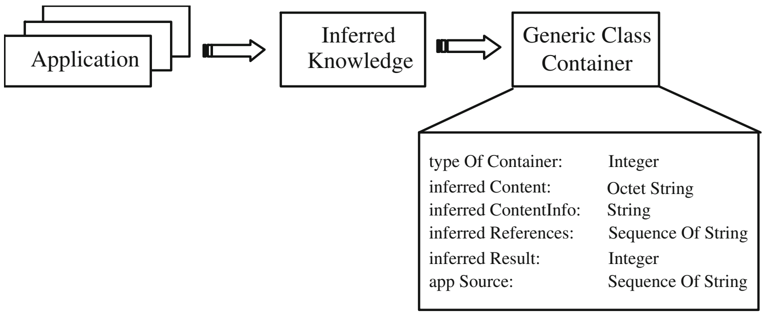

Application
Inferred Knowledge
Generic Class Container
type Of Container: Integer
inferred Content: Octet String
inferred ContentInfo: String
inferred References: Sequence Of String
inferred Result: Integer
app Source: Sequence Of String

Fig. 2 The DEN-ng approach for representing inferred knowledge

conjunction with the inferredContentInfo attribute; this attribute is a string, and describes how to interpret the data stored in the inferredContent OctetString. The inferredReferences attribute is an array of strings, one for each reference to an external concept that is required to use this knowledge (e.g., a particular lexicon might be required to interpret the result of the inference operation). The inferredResult is an enumerated integer, and defines one of a set of standard result codes that can be used so that other applications that cannot understand the inferred data can substitute that data with an equivalent result. Finally, the appSource attribute is an array of strings, where each string defines a unique identifier that in turn identifies an application that produced the inferred data. This is useful for tracing the results of the inference operation in case it does not agree with other data.

Strassner [7] defined the concept of a Policy Continuum and [23] reported on its implementation; [24] then formalized this model. The Knowledge Continuum was defined by extending this work and applying it to knowledge [25]. This provides a mechanism to establish continuity between otherwise disparate viewpoints of knowledge.

Similar to the Policy Continuum, the Knowledge Continuum asserts that in order to ensure the correct understanding of knowledge at one abstraction, and to be able to relate that abstraction to other views, knowledge itself must be represented in a series of views, where each view has meaning in a specific frame of reference, and where each successive view is generated from a transformation being applied to the preceding view. As in the Policy Continuum, the Knowledge Continuum removes the differentiation of transformation from refinement, resulting in a pure transformation pipeline. This is a formalization of the adaptive pipeline pattern described in [26], which is a forward engineering approach in which any given source model is first restructured in the transformation pipeline according to a formal language and formal transformation theory; this is required in order to prove that the transformation(s) performed preserve the semantics of the models. Transformations are then applied to the resulting formal language. A conceptual view of the Knowledge Continuum is shown in Fig. 3.

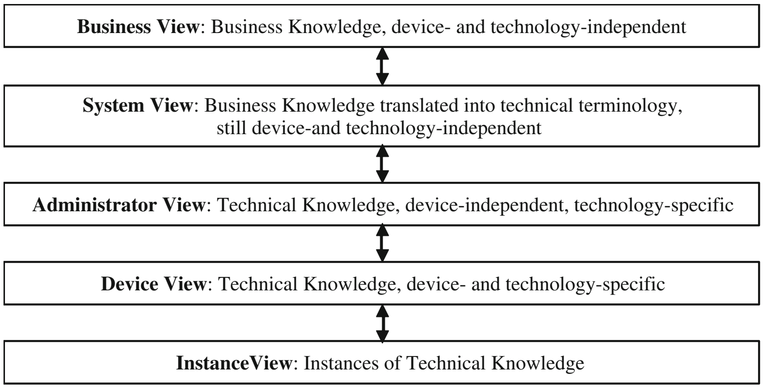

Business View: Business Knowledge, device- and technology-independent
System View: Business Knowledge translated into technical terminology, still device-and technology-independent
Administrator View: Technical Knowledge, device-independent, technology-specific
Device View: Technical Knowledge, device- and technology-specific
InstanceView: Instances of Technical Knowledge

Fig. 3 The Knowledge Continuum

Knowledge is assigned to a particular level in the Knowledge Continuum based on whether it is business or technical in nature, and whether it is device- and technology-specific or not.

5 DEN-ng Context-Aware Policy Model

The DEN-ng model is built on two important concepts: patterns and roles. This formalism, along with best current practices such as using roles to abstract concepts, is the source of much of the inherent extensibility of the model. For example, the use of roles makes a design inherently scalable by abstracting individual users, devices, and services into roles that can be played by various managed entities. DEN-ng is unique, in that roles are not limited to just people; rather, they may include roles representing resources, services, products, locations, and other managed entities of interest.

5.1 Integrating Models and Ontologies

Networks are made up of heterogeneous devices, each with its own vendor-specific concepts and implementation dependencies. Hence, incompatibilities between different means of representing and processing information, along with defining the meaning of said information, arise. These issues result in a phenomenon called cognitive dissonance. Cognitive dissonance [27] results in a situation where two supportable, held beliefs are in opposition to each other. In terms of UML models, this most often results when considering instances of models due to the lack of precision in the specification of relationships in the presence of inheritance hierarchies. Furthermore, UML does not contain the constructs necessary to support the definition of knowledge or reasoning about knowledge, which requires formal semantic definitions that enable unambiguous representations and organization of the representations that facilitate the calculation of semantic similarity construction [25, 28]. As another example, UML does not have the mechanisms required to establish semantic relationships between information (e.g., synonyms and antonyms). For example, there is no ability in UML to represent explicit semantics (i.e., the definitions used in UML are implicit). In addition, relationships, such as “is similar to”, that are needed to relate different vocabularies (such as commands from different vendors) to each other, along with tense, modals, and episodic sequences, cannot be defined.

Thus, when different terms need to be equated, existing standards-based models and vendor programming commands, such as SNMP and vendor-specific command-line interfaces, express knowledge differently. This is shown conceptually in Fig. 4.

These limitations also make it impossible to do basic functions, such as define the set of simple network management protocol (SNMP) monitoring commands that can determine if a command line interface (CLI) configuration command was successful or not.

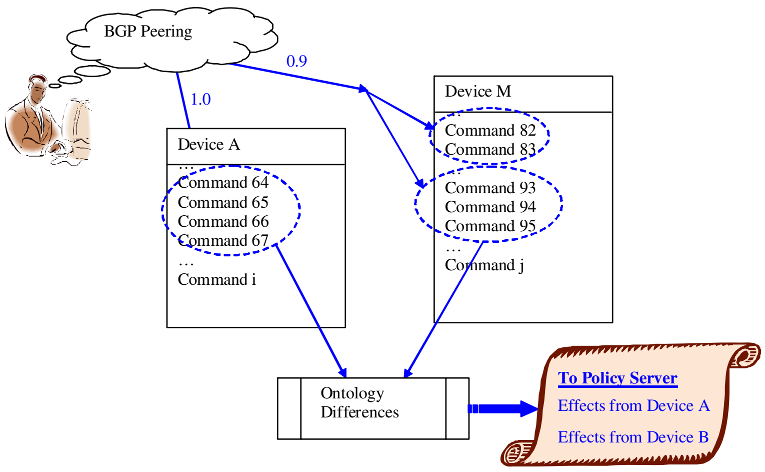


Fig. 4 Mapping a high-level function to different command sets

5.2 Harmonizing Management Data

Given the diversity in vendor-specific management data and technologies, we propose an iterative approach to identifying and organizing management data received by sensors and inferred by learning and reasoning algorithms. This process is based on first, using the DEN-ng information model as a template to match received sensor data and infer relationships; second, construct semantic relationships to augment the meaning of identified modeled data with ontological data; third, use the semantic relationships to identify existing and/or construct new model elements (e.g., attributes and relationships) in the information model; fourth, repeat until an acceptable confidence level is achieved.

The above process provides a methodology to not just use data from an existing knowledge base, but more importantly, to *continually check and update* the knowledge base.

5.3 Our Dynamic Knowledge Base

Figure 5 shows the first step in constructing our knowledge base: building the foundation using modeled data.

Step 1 constructs an information model. Note that the majority of current network management approaches do not use an information model per se; most use a set of data models that may be, but usually are not, related to each other. Our approach uses a common information model to define common concepts from which data models can be derived. We map existing data models to DEN-ng using reverse engineering [29]. Accordingly, step 2 derives one or more data models from this information model (in order to accommodate the needs of different management applications that use different protocols and data structures, as well as to model vendor-specific data). Step 3 constructs a system object inventory, whose overall

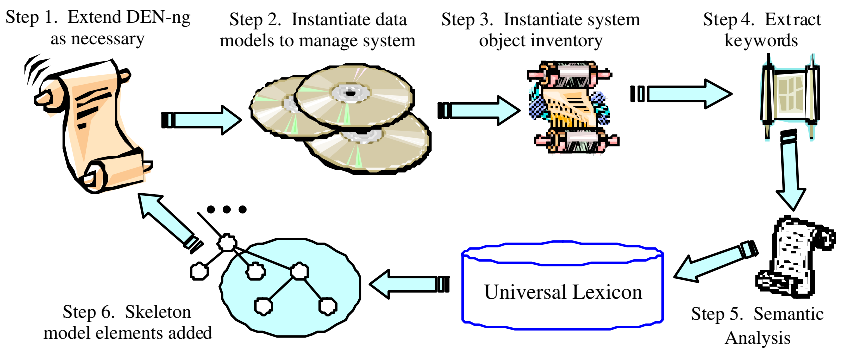

Step 1. Extend DEN-ng as necessary
Step 2. Instantiate data models to manage system
Step 3. Instantiate system object inventory
Step 4. Extract keywords
Step 5. Semantic Analysis
Universal Lexicon
Step 6. Skeleton model elements added

Fig. 5 Building the foundation using modeled data

structure is defined by the DEN-ng information model, and hence is independent of the platform, language, and protocol specifics of each data model. This simple but novel step uses the information model as a design guide as well as an aid in organizing runtime objects that are indirectly instantiated from it. This enables the different ManagedEntities (such as services and resources) and their associations to be used as nodes and edges in the state machines of DEN-ng, which is how services and resources are managed. From a knowledge engineering point of view, these objects also represent key concepts that, when used with state automata, reflect current and/or desired system behavior. This is a model-driven approach that emphasizes the use of models not just for analysis and design, but for all facets of network and system management. A management system using this approach can update its knowledge base dynamically by looking for changes observed in the structure and content of its knowledge during runtime and update the models accordingly. Step 4 examines the instantiated objects and extracts keywords and phrases from this object inventory in order to determine, at runtime, if (1) extracted data agrees with the model of that data, and (2) there is additional "hidden" or new knowledge that was not modeled. Step 5 analyses these keywords and phrases using a variety of techniques, ranging from simple pattern matching to using ontologies to determine the cognitive similarity between concepts. Step 6 constructs graphs that represent the knowledge extracted from the models.

Figure 6 shows the second step in constructing our knowledge base: augmenting the foundation of our knowledge base with additional concepts and meaning that are extracted from applicable ontologies.

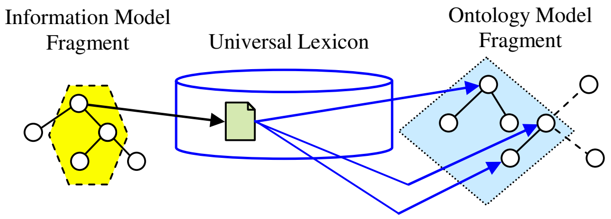


Fig. 6 Augmenting modeled data with ontological meaning

Figure 6 shows the general case of mapping an information model fragment to a set of ontologies using a set of terms in our lexicon. Each of the arrows emanating from the terms in the lexicon represents one or more different linguistic relationships (e.g., synonyms, antonyms, homonyms, and custom relationships, such as 'is similar to'). Note that in general, this method will identify a part of a taxonomy without identifying all concepts in the taxonomy; this is represented by the two dashed lines in the ontology model fragment (one being the superclass of the two concepts found, and the other being a subclass that was not found).

Each model element could potentially have more than one meaning. In a well-designed model, such as one based on DEN-ng, this can be minimized by abstracting functionality using roles, which codify specific functions. For example, instead of using the generic term "router", a system can use roles to identify the type of router function that is being performed (e.g., interfaces that interact with core routers vs. those that interact with edge routers, or roles that define the overall function of a router, such as "border router" vs. "transit router" vs. "edge router"). Metadata can also be used to help disambiguate the function of a model element. For those cases where it is impossible to distinguish the exact function of a model element, a semantic analysis must be done to establish the particular meaning that should be used in this context. This means that a set of (possibly different) semantic mappings must be performed for each meaning for each model element. If a term has multiple valid meanings, one semantic relationship is defined for each distinct meaning.

However, before this potentially expensive set of operations is performed, an alternative exists: examine the information model to find a set of model elements that are related to the model element whose meaning is ambiguous, and treat this group of model elements as a complete concept to search on. To use an analogy, an individual model element can be viewed as a noun, whilst a group of model elements can be viewed as a phrase. Just as in English sentences, where a phrase gives additional context to each of its components, analyzing a group of model elements gives additional context to each of the contained model elements. Optionally, our approach allows the use of a programmable threshold that rejects mappings that do not have a high enough value of semantic equivalence.

Essentially, the above semantic resolution process compares the meaning (i.e., not just the definition, but also the structural relationships, attributes, etc.) of each element in the first sub-graph with all elements in the second sub-graph, trying to find the closest language element or elements that match the semantics of the element(s) in the first sub-graph. Often, an exact match is not possible; hence, the semantic resolution process provides a ratioed result, enabling each match to be ranked in order of best approximating the collective meaning of the first sub-graph. As a consequence, the number of elements in the second sub-graph, and the overall structure of the second sub-graph, do not have to be the same (or even similar) to the number of elements and structure of the first sub-graph. Wong et al. [28] provides an example of such a semantic similarity algorithm.

The next step is illustrated in Fig. 7. In this step, each ontology concept that was identified in the semantic matching process is now examined to see if it is related to other concepts in the same or other ontologies. As each new concept is found, it is

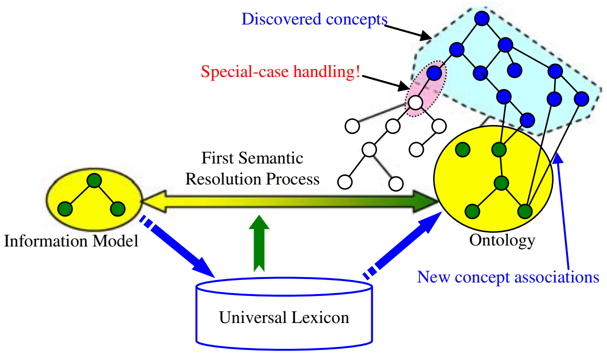


Fig. 7 Adding new ontological concepts

marked for possible addition to the existing concepts that were already matched from the lexicon. The newly added concept is then checked to see if it is related to any of the terms identified in the lexicon. If it is, the new concept is added; this is shown in the dashed polygon in Fig. 7. If it is not directly related to a term in the lexicon (as shown in the dotted ellipse in Fig. 7), then additional checks are formed to see if (1) it is indirectly related, or (2) it should be added as a new relation (i.e., this represents new knowledge).

The underlying idea for these additional checks is to verify that each new concept reinforces or adds additional support for the concept that was already selected. Hence, this process can be thought of as *strengthening the semantics* of the match.

These new semantic associations, along with the new concepts discovered in the ontology, can now be used to find new model elements. Each new ontology concept is first mapped to one or more terms in the lexicon, and then each of those terms are mapped to model elements. As before, the algorithm attempts to match groups of related concepts to groups of related model elements. This has the effect of increasing the semantic similarity between two concepts; as larger groups of concepts are matched to larger groups of model elements, a stronger correlation between the meaning of the grouped concept and the group of facts is established. This is, in effect, a self-check of the correctness of the mapping, and is used to eliminate concepts and model elements that match each other, but are not related to the managed entity that is being modeled. This is shown in Fig. 8.

During this process, any new or changed knowledge is dynamically uploaded to the system's knowledge base. Hence, new concepts that have semantic meaning that are discovered from managing the system can be dynamically added to the information model, meaning that the knowledge base of the system dynamically changes to reflect new knowledge learned through experience. These steps are described in more detail in [25].

5.4 The DEN-ng Context Model

In order to accommodate as flexible a set of definitions as possible, the core of our context design consists of two classes, Context and ContextData, as shown in Fig. 9.

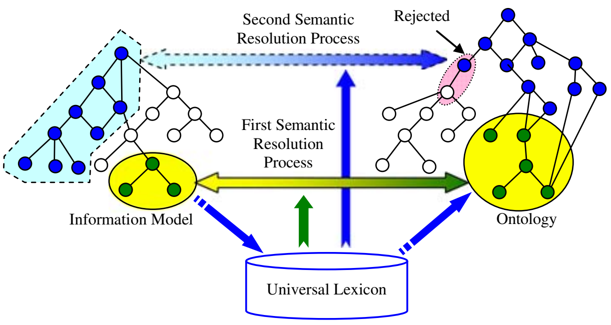

Second Semantic Resolution Process
Rejected
First Semantic Resolution Process
Information Model
Ontology
Universal Lexicon

Fig. 8 Semantic resolution process

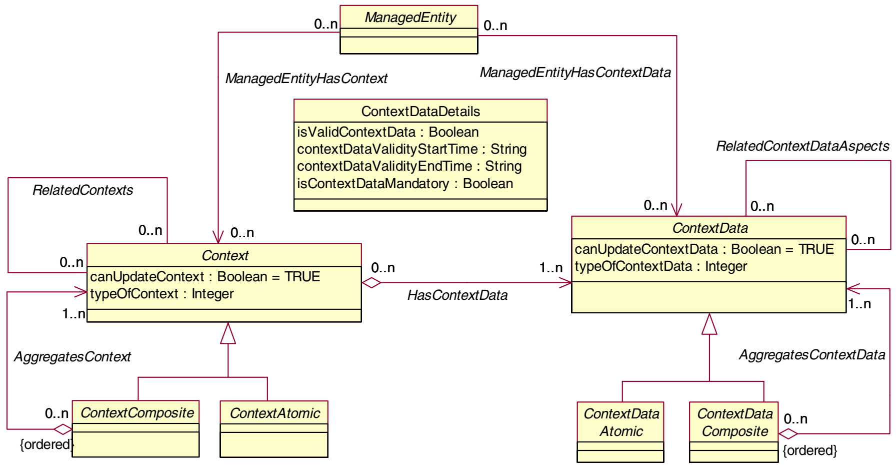

ManagedEntity
0..n
0..n
ManagedEntityHasContext
ManagedEntityHasContextData
ContextDataDetails
isValidContextData : Boolean
contextDataValidityStartTime : String
contextDataValidityEndTime : String
isContextDataMandatory : Boolean
RelatedContexts
RelatedContextDataAspects
Context
canUpdateContext : Boolean = TRUE
typeOfContext : Integer
ContextData
canUpdateContextData : Boolean = TRUE
typeOfContextData : Integer
HasContextData
0..n
1..n
AggregatesContext
AggregatesContextData
ContextComposite
ContextAtomic
ContextData Atomic
ContextData Composite
{ordered}
{ordered}

Fig. 9 Core of the DEN-ng context model

The DEN-ng model enables context to consist of multiple distinct sets of related data and knowledge. Therefore, we use two different classes to represent context. The Context class is used to represent a completely assembled representation of context, while the ContextData class is used when context contains multiple distinct types of different data that need to be combined in order to determine the overall context of an entity. For example, when modeling a phone call which can involve handover between two different technologies, we instantiate two different sets of "sub-contexts"—each consists of a collection of ContextData classes that is bound to a particular technology. This enables us to better manage the phone call, since the underlying technologies are themselves fundamentally different. More importantly, this design approach discourages the construction of a single context model, which is usually bound to a single set of applications, and instead encourages building a context model from a set of *reusable* component models; each of the component models represents a particular set of aspects of context, which enables it to be more easily re-purposed.

In FOCALE, we implement the above notion in a reusable manner by providing a generic mechanism for associating detailed models and ontologies with content. Consider the concept of location. Instead of defining a "locatedAt" attribute in the Context class (which would necessarily have a fixed meaning associated with it), we can have the Context class reference a set of location classes that provide more detailed information and semantics for what location "means" to the Context of this particular ManagedEntity. This also allows the semantics of location to change as a function of context. This flexibility is unique in existing context models. Figure 10 shows how this approach works in practice.

First, we define a ContextData to represent the concept of location. We link the ContextData class to a standard set of location classes as defined in the DEN-ng model, which describe location in a reusable way. Thus, we reuse the basic location model, and then attach semantics of the location through the ContextData classes. Note that location is more than just latitude and longitude, but also includes height, proximity, co-location with other entities, and orientation. This genericity is why a set of classes (and appropriate units of measurement in 2-D or 3-D space) are required for defining location, as opposed to a simple attribute. In particular, this enables other applications that use the concept of location to reuse not just the set of classes describing the location, but the ContextData classes that are linked to it. In this way, we create reusable context models that are application-independent.

The three classes ContextDataFact, ContextDataInference, and ContextDataSemantics are each containers that external applications can use to populate computed and/or inferred values. The containers which reasoning applications use are defined as was previously shown in Fig. 2. The association ContextDataHasLocationSemantics is used to pull location information in to be analyzed; ContextDataFacts and ContextDataInferences are then produced describing the location. For example, the GPS coordinates of two people in the same city can be compared to compute the proximity of one to the other. As another example, inference can be used to determine which people trapped in a building that has a fire are in greater danger.

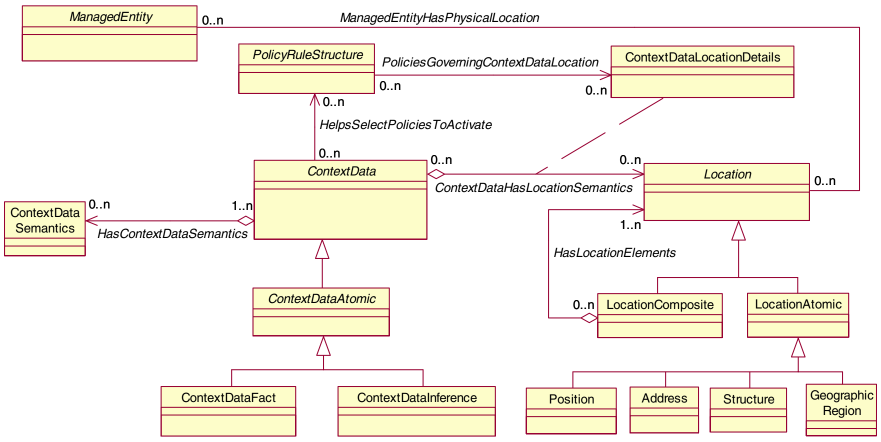


Fig. 10 Example of ContextData for location

The DEN-ng ContextData taxonomy is divided into a domain-independent portion (its “Upper Ontology”) and a set of domain-specific portions, as shown in Fig. 11. The upper ontology captures generic context knowledge, while each domain-specific ontology delineates the details of concepts defined in the upper ontology in a manner specific to each sub-domain. This enables reusable context models to be created by attaching application-specific ContextData Domain Specific Ontology data to ContextData Upper Ontology information. This separation of common vs. domain-specific semantics reduces the burden of context processing for each individual application, and ensures simple re-purposing through its ability to dynamically connect or disconnect different ontologies to or from the upper ontology in order to meet the needs of resource-constrained devices, such as a cell phone. This latter is an important design consideration. For example, when a user leaves his home to drive to work, our system can swap out the “home context” ontology with a “car context” ontology that reflects the different devices and their capabilities that the user now has access to. Similarly, when the user arrives at work, the “car context” ontology is swapped out with the “work context” ontology. Each context model is reusable, and from a processing point-of-view, each model only needs to be loaded when the context changes to make it relevant. This streamlined processing is critical for applications such as those embedded in cell phones, since they do not have a lot of freely available processing power.

Our Upper Ontology consists of five top-level types of information: Managed-Entity, Location, Time, Activity, and PersonOrGroup. Each of these information types is detailed in the DEN-ng information model.

Both the Context and ContextData classes use the composite pattern for flexibility and extensibility. The ContextAtomic and ContextDataAtomic classes

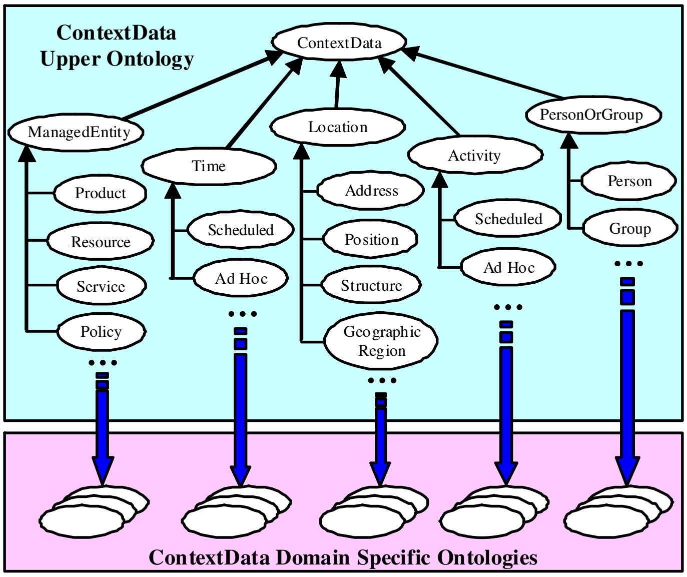


Fig. 11 DEN-ng ContextData taxonomy

represent context that can be modeled as a single, stand-alone object. In contrast, the ContextComposite and ContextDataComposite classes represent context objects that are composite in nature (e.g., made up of multiple distinct Context or ContextData objects that can each be separately managed). This enables hierarchies of context information to be related to other hierarchies of context information. Each context node (a Context object) can have a set of ContextData objects that provide further detail describing the characteristics and behavior of that node. For example, the Context object "Communication" could have the following ContextData objects associated with it: PSTN (to model the characteristics of fixed telephone lines), CellularDevice (to model the characteristics of mobile phones and PDAs), ComputerDevice (to model the characteristics of laptops and desktops) and VisualAudioDevice (to model the characteristics of a television with Internet capability). Each of these four classes of device uses different types of media and provides different types of communication experiences. This model is especially useful to model a user that has access to one or more of these devices at any particular time; in this case, the model defines how the user can communicate.

The purpose of the ContextDataDetails association class is to define the particular semantics of how ContextData relates to Context. This enables different types of ContextData, each modeling a specific aspect of an overall Context, to be aggregated together with their own semantics.

The ContextDataFact and ContextDataInference classes are used to represent additional data that is either known a priori or can be inferred from other knowledge about a given ContextData. For example, a given access point's signal strength can vary over time—this can be observed by a sensor and a decision made as to whether the access point is stable enough to support mission-critical data communication over an encrypted link or not. Similar capabilities exist for the Context class—it has ContextFact and ContextInference subclasses.

The ContextSemantics and ContextDataSemantics classes represent data and/or knowledge that describe the behavioral aspects of the Context and ContextData objects, respectively, that this ManagedEntity are associated with, and represent a convenient point for *fusing* information from ontologies with data from information and data models. This done by using these points to associate code generated from the model with code generated from the ontologies through the use of the intelligent containers shown in Fig. 2. This in turn enables modeled data, which represent facts, to be augmented with additional semantics from ontological data; this combination forms an efficient and self-describing set of knowledge that can be fed into machine-based learning and reasoning systems [25]. They also present convenient points for either augmenting context information (e.g., tagging it with metadata to enhance information retrieval) and/or using context data to perform (for example) a set of services (which is the subject of Sect. 5.5). In addition, they enable new knowledge to be dynamically incorporated into the definition of context using the methodology of Sect. 5.3. Finally, these two semantics classes enable the application to declare what it needs to complete its view of context, as opposed to merely obtaining context information. However, these latter examples are each complex processes and beyond the scope of this paper.

5.5 DEN-ng Context-Aware Policy Model

Strassner [7] defines policy as "Policy is a set of rules that are used to manage and control the changing and/or maintaining of the state of one or more managed objects." In FOCALE, context is related to the state of an entity. This is shown in Fig. 12.

State can vary from object type to object type. For example, for a person, it can refer to the activity that the person is currently doing, physiological factors such as whether the person is sick or tired, whether the person is busy or not, different physical conditions, and other factors. In contrast, the state of a network device includes its operational status, its power state, and other attributes that can be queried. In particular, changes in context trigger state transitions that adjust the behavior the entity in accordance with the changes in the environment that it exists in. For example, imagine a user is switching between a business profile and an entertainment profile. The former uses a special service provider that offers encrypted, highly secure communication for business use, while the latter uses a completely different set of service providers (one for local and a different one for long distance calling) as well as links to social networks. In this situation, the policies defined as being usable for that context state will in general be different: some policies may be the same (e.g., rules about which devices can be used), some policies may be completely different (e.g., rules about which services can be used), and some policies may be modified (e.g., rules that govern communication).

The DEN-ng model [30] represents three different types of policy rules in order to provide context-aware systems with sufficient flexibility to define rules. The PolicyRuleStructure class is the superclass of different types of policy rules; this paper concentrates on one type, known as an ECA (event-condition-action) policy, and represents the structure of a policy rule. Similarly, the PolicyRuleComponent-Structure class is the superclass for PolicyEvents, PolicyConditions, and PolicyActions. This enables different types of Policy Rules to have different structural components. This is shown in Fig. 13.

Metadata is modeled as a separate hierarchy, and is shown in Fig. 14.

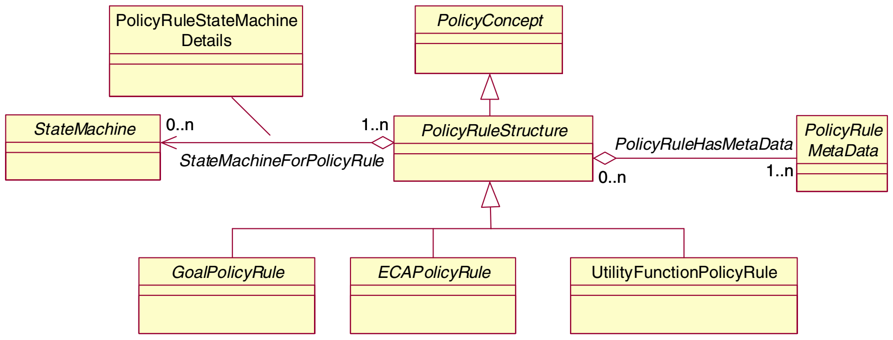


Fig. 12 Policy affects state

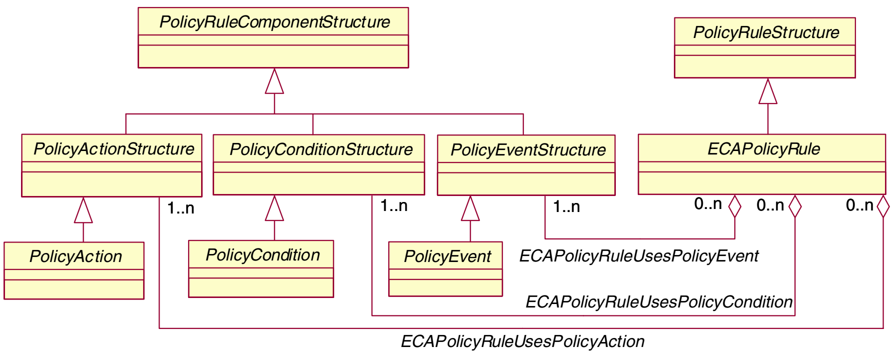


Fig. 13 The ECAPolicyRule class

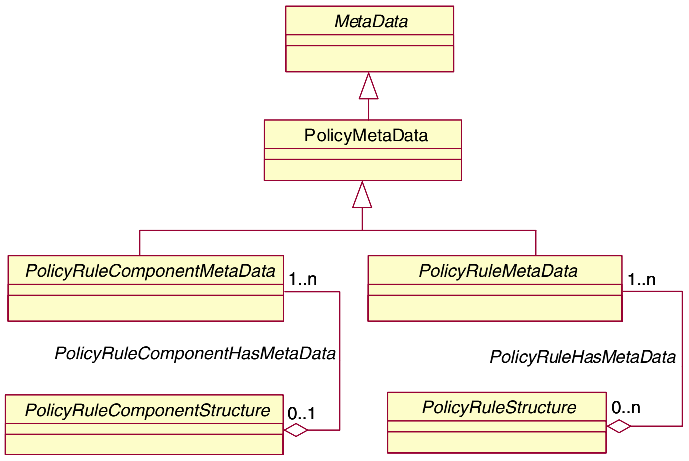


Fig. 14 Policy metadata hierarchy

The PolicyMetaData class defines metadata that applies to different types of Policies, such as (but not limited to) ECAPolicies. This decouples common metadata that different Policy representation systems need from the actual realization of the Policy, enabling both ECA- and non-ECA Policies to use the metadata contained in this class. It also decouples the representation and structure of a particular type of policy (e.g., an ECAPolicy) from the metadata. This is critical for properly constructing ontologies from policy models.

Figure 15 shows conceptually how context is used to affect policy [31]. The SelectsPoliciesToActivate aggregation defines a set of Policies that should be loaded and activated based on the current context. Hence, as context changes, policy can change accordingly, enabling our system to adapt to changing demands. Note that this selection is an “intelligent decision”, in that the selection process depends on other components that are part of a particular context. The PolicyResultAffects-Context association enables policy results to influence Context via the ContextCon-trollerComponent, the application that manages Context. For example, if a policy

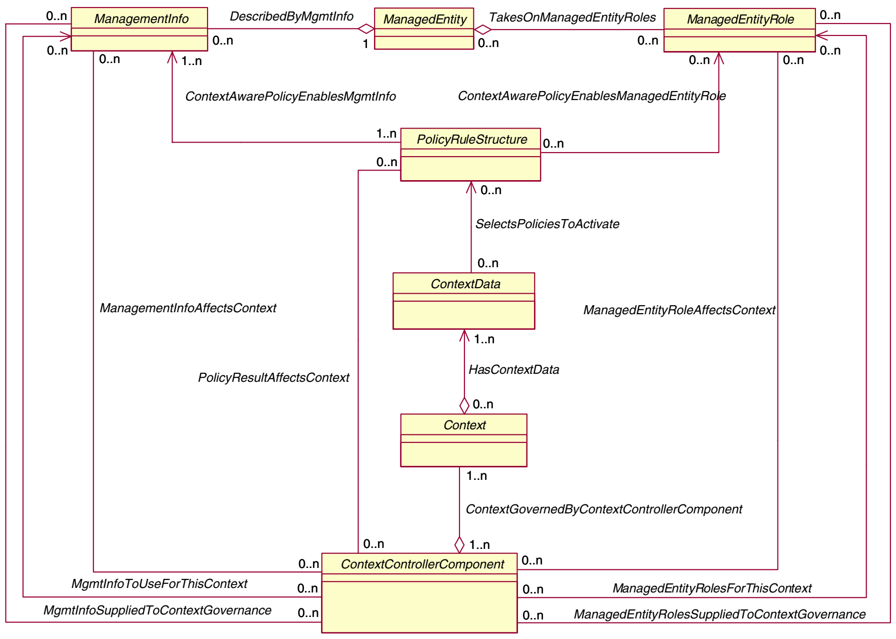

ManagementInfo
DescribedByMgmtInfo
ManagedEntity
TakesOnManagedEntityRoles
ManagedEntityRole
ContextAwarePolicyEnablesMgmtInfo
ContextAwarePolicyEnablesManagedEntityRole
PolicyRuleStructure
SelectsPoliciesToActivate
ContextData
ManagementInfoAffectsContext
ManagedEntityRoleAffectsContext
HasContextData
PolicyResultAffectsContext
Context
ContextGovernedByContextControllerComponent
ContextControllerComponent
MgmtInfoToUseForThisContext
MgmtInfoSuppliedToContextGovernance
ManagedEntityRolesForThisContext
ManagedEntityRolesSuppliedToContextGovernance
0..n
1..n
1

Fig. 15 Conceptual DEN-ng context-aware policy model

execution fails, not only did the desired state change not occur, but the context may have changed as well.

The selected working set of Policies uses the ContextAwarePolicyEnablesManagedEntityRoles association to define and enable the appropriate ManagedEntity roles that are influenced by this Context; each ManagedEntityRole defines functionality that the ManagedEntity can use. In this way, policy indirectly (through the use of roles) controls the functionality of the system, again as a function of context. Similarly, ContextAwarePolicyEnablesMgmtInfo defines the set of management data that is useful for this Context; ManagementInfoAffects-Context represents feedback from these management data regarding the execution of the policy rule. Once the management information is defined, then the two associations MgmtInfoAffectsContext and ManagedEntityRoleAffectsContext codify these dependencies (e.g., context defines the management information to monitor, and the values of these management data affect context, respectively).

Finally, the ContextControllerComponent defines its own set of ManagedEntityRoles and ManagementInfo to use to monitor the environment; feedback from the ManagedEntities identified by their roles and specific measurements (in the form of ManagementInfo) are used by the ContextControllerComponent to operate its own finite state machine (FSM). This FSM is used to orchestrate the actions of the ContextControllerComponent, including which ManagedEntities it should determine the context for, how a particular element of context should be analyzed, and what procedures for determining its context should be used.

5.6 Realizing Context Awareness

The challenge of modeling context is how to capture, process, and exploit it to provide the correct behavior in the correct form to the correct user at the correct time in the correct place [22]. Context models are often associated with pre-defined rules [32], as is the case with FOCALE [15]. In the case of FOCALE, policy rules are used to specify the set of actions to take based upon one or more changes in context. For example, if a dependency exists between the location of a person and the type of communications used, then a change in location can be used to determine a different access mode, or modulation, or use of frequency, or other changes to the communication used. This is shown in Fig. 10, where the HelpsSelectPolicies-ToActivate association determines the set of policy rules to be used given a change in context via the ContextDataLocationDetails association class, which helps determine the specific semantics of the aggregation relating Location to Context (ContextDataHasLocationSemantics).

5.7 Strengthening Semantics

The discussion in Sect. 5.3 above showed how our combination of modeled and ontological data can be used to *strengthen* the semantics being inferred. This in effect provides a self-check of the consistency and integrity of the context information that has been collected. This is especially important when multiple contextual data from different sources needs to be integrated. Since data from different sources can have different qualities (e.g., accuracy, certainty, and freshness) as well as different formats, we have two important problems to solve: (1) the harmonization of different qualities of data, and (2) the alignment of different ontologies to facilitate extraction of diverse data. We propose a programmable threshold for the former, which enables us to weight the contribution of different data sources (but see the future work section for desired embellishments), and the use of standard plus custom semantic equivalence relationships for the latter.

Coutaz et al. [33] points out that context is not simply a state but part of a process. In other words, the system must behave correctly during the entire process. The DEN-ng model in Fig. 12 is used as shown in Fig. 16 to relate state to an activity, which is in monitored by the autonomic manager. Changes in the state trigger analysis using the combination of modeled and ontological data, which is then related to the appropriate set of state machines. These use context to identify the set of policies that should be executed, which reconfigure the system appropriately. The monitoring processes then check the result to ensure that it was as expected.

There are three approaches to consider. *Ontology merging* constructs a single ontology from two or more different ontologies related to the same subject. This is called *merging* because the constructed ontology includes data from all source ontologies. *Ontology integration* builds a single ontology in one domain from two or more ontologies in different domains. *Ontology alignment*, also called *ontology mapping*, constructs links between the different source ontologies, but keeps the

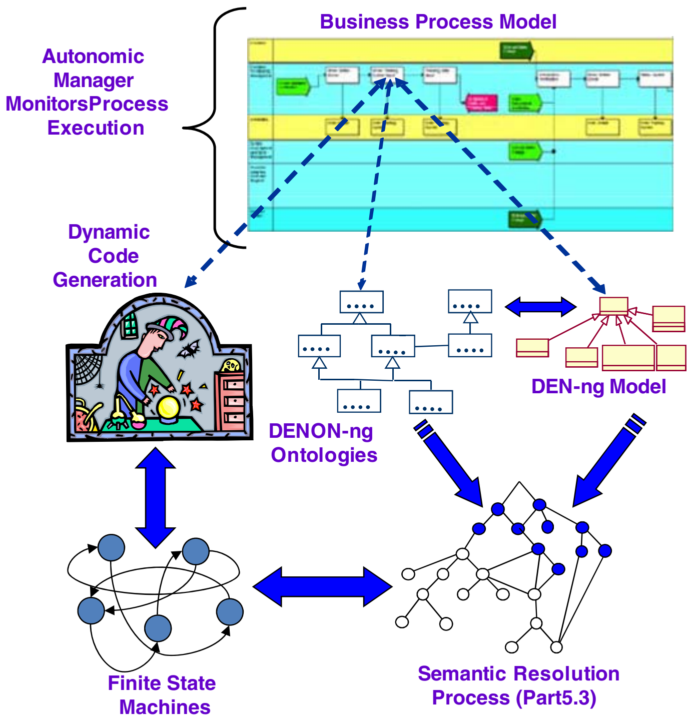


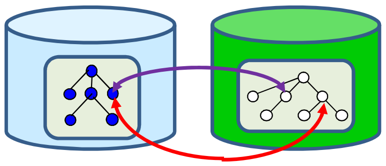
Fig. 16 Relationship between state and process

source ontologies unchanged. We use this approach, since our ultimate aim is to be able to reuse ontologies. This has been discussed in detail in (see, for example, [34, 35]). It is conceptually shown in Fig. 17.

The two lines from the node on the left schema represent two mappings to two different nodes in the right schema. This is exemplary, but is indicative of a number of problems that need to be solved, including: (1) how to detect and eliminate duplicate entries due to taxonomical, data structure, and other differences; (2) consolidation of different entries, where each entry contains data that the other entry doesn't contain; (3) lack of sufficient expressivity in a node or set of nodes in a source schema to map it with a high enough probability to a node or set of nodes in a target schema; (4) determining that knowledge is missing.

Several candidate matching algorithms can be applied to support the determination of semantic mappings. For example, [25] uses *semantic matching* to solve these problems. The simplest version of this approach involves the following steps:

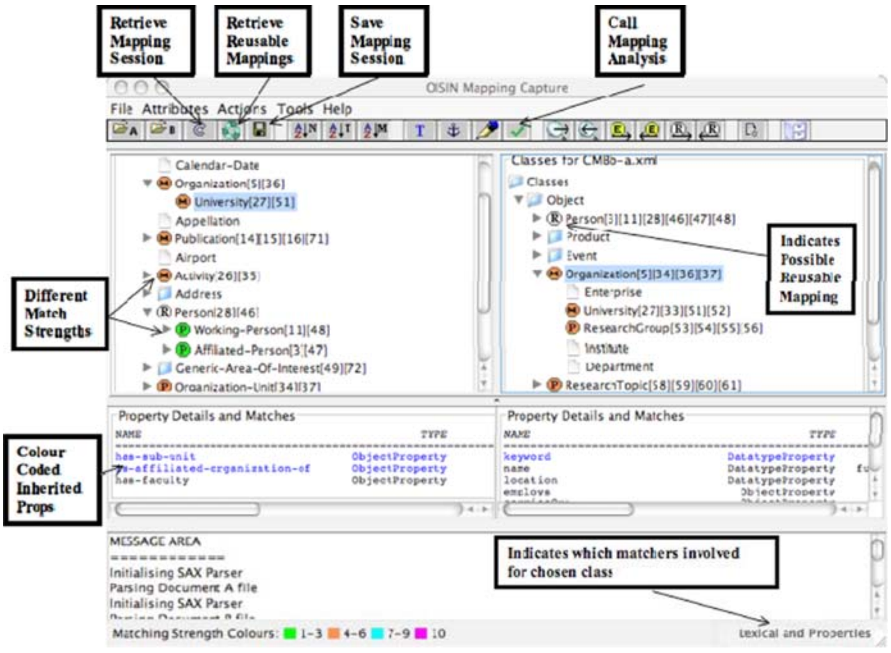
Fig. 17 Ontology alignment



1. Transform the ontologies to be matched into equivalent XML schemata
2. Parse each node into a set of tokens
3. For each node, find the strongest match using either (1) pattern matching of modeled and/or ontological data and/or (2) linguistic matching by comparing the tokens according to the order {equality, superset, subset, overlap, disjoint} ([28] provides an example of such an algorithm)
4. For matching nodes, repeat the above for each node attribute
5. Repeat this approach for groups of nodes, as described in Figs. 5, 6, 7, and 8; this strengthens the results of the matching process.

In FOCALE, another alternative for matching includes the use of custom relationships, such as "has same effect as". In addition, a hybrid reinforcement learning loop is used to observe the above processes to help fine-tune the matching process. An example of a hybrid approach is the OISIN system [36], shown in Fig. 18, which uses multiple semantic models in the domain of research to perform the mapping.

According to the Shvaiko and Euzenat classification [35], this incorporates a language-based matcher, a linguistic resource matcher, a type-based matcher, and a reuse-based matcher, in combination to support a user or an application in their determination of semantic mappings. For example, this system has been successfully demonstrated in mapping between the Common Information Model (CIM) and some Simple Network Management Protocol (SNMP) models [37]. Two SMI information models (MIBs), the ENTITY-MIB and HOST-RESOURCES-MIB,


Retrieve Mapping Session
Retrieve Reusable Mappings
Save Mapping Session
Call Mapping Analysis
OISIN Mapping Capture
File Attributes Actions Tools Help
Calendar-Date
Organization[5][36]
University[27][51]
Appellation
Publication[14][15][16][71]
Airport
Activity[26][35]
Address
Person[28][46]
Working-Person[11][48]
Affiliated-Person[3][47]
Generic-Area-Of-Interest[49][72]
Organization-Unit[34][37]
Classes for CMBb-a.xml
Classes
Object
Person[3][11][28][46][47][48]
Product
Event
Organization[5][34][36][37]
Enterprise
University[27][33][51][52]
ResearchGroup[53][54][55][56]
Institute
Department
ResearchTopic[58][59][60][61]
Different Match Strengths
Indicates Possible Reusable Mapping
Colour Coded Inherited Props
Indicates which matchers involved for chosen class
Matching Strength Colours: 🟩 1-3 🟧 4-6 🟦 7-9 🟪 10
Lexical and Properties

Property Details and Matches

```
NAME                                TYPE
---------------------------------------------
has-sub-unit                        ObjectProperty
has-affiliated-organization-of      ObjectProperty
has-faculty                         ObjectProperty
```

Property Details and Matches

```
NAME                                TYPE
---------------------------------------------
keyword                             DatatypeProperty
name                                DatatypeProperty
location                            DatatypeProperty
employs                             ObjectProperty
```

```
MESSAGE AREA
============
Initialising SAX Parser
Parsing Document A file
Initialising SAX Parser
Parsing Document B file
```

Fig. 18 OISIN semantic mapping tool

were chosen to be semantically mapped to parts of the CIM object model. These models have the advantage of containing many common concepts expressed in different ways. Furthermore, the CIM model contains a number of *mappingStrings* attributes containing semantic information about some parts of the CIM model in terms of roughly equivalent SMI concepts. In order for ontological mapping to be performed on the models, they each were first translated in an ontological format. The mappings were then used to allow network managers to express interest in certain events that were being distributed on an event based network. As a concrete example, a manager expressed interest in receiving information about devices, in the form of a CIM query (e.g., *CIM_Printer*). By exploiting the ontological mappings, this query was automatically satisfied by SNMP events related to devices described as *hrPrinterEntry*, described in the SMI HOST-RESOURCES-MIB. Although successful in its demonstration of the use of general purpose semantic mapping tools within the network management domain, some key issues were raised that still need to be noted and addressed into the future: (1) A combination of matching algorithms are advocated as pure lexical matching approaches alone are problematic due to the wide variety of modeling practice of standards bodies in naming entities through concatenation of strings; (2) the complexity of scale in presenting users with numerous possible matches from which they need to determine mappings.

6 Mapping to Business Concerns

The concept of the Policy Continuum [7] was originally defined to serve as a bridge between different constituencies that used policies to manage and configure network devices, but used different concepts and terminologies that were specific to different constituencies (e.g., business analysts and product managers vs. programmers vs. network administrators). Each level in the Policy Continuum addresses a specific set of users that has a very specific understanding of the managed entities operating at that particular level of abstraction. Davy et al. [24] showed how to map different concepts within the same continuum level to each other. For example, the business user wants Service Level Agreement (SLA) information, such as revenue and money for violating the SLA, and is not interested in the type of queuing that will be used to condition traffic corresponding to that particular SLA. Conversely, the network administrator may want to develop CLI commands to program the device, and may need to have a completely different representation of the policy in order to develop the queuing CLI commands. The objective of the Policy Continuum is to define a set of transformations that relate different abstractions of the same concept to each other. In this example, the business notions of “revenue” and “cost” need to be related to traffic classification and conditioning (e.g., dropping, queuing, scheduling, etc.). Information models do not provide such a mapping. Our approach uses the combination of modeled and ontological data to infer such relationships. For example, Fig. 19 shows an extract of a simplified DEN-ng mapping of contract entities, such as an SLA, to two different types of services.

A customer-facing service is one that is directly visible to a customer, while a resource-facing service is one that is required by the customer-facing service to

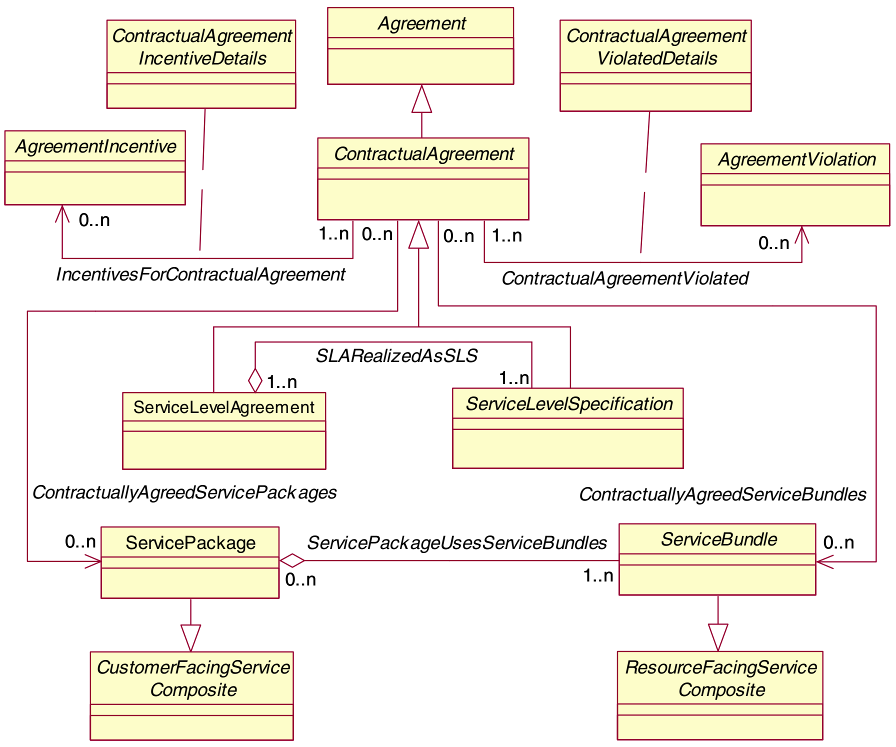

ContractualAgreement IncentiveDetails
Agreement
ContractualAgreement ViolatedDetails
AgreementIncentive
ContractualAgreement
AgreementViolation
0..n
1..n
0..n
0..n
1..n
0..n
IncentivesForContractualAgreement
ContractualAgreementViolated
SLARealizedAsSLS
1..n
1..n
ServiceLevelAgreement
ServiceLevelSpecification
ContractuallyAgreedServicePackages
ContractuallyAgreedServiceBundles
0..n
ServicePackage
ServicePackageUsesServiceBundles
ServiceBundle
0..n
0..n
1..n
CustomerFacingService Composite
ResourceFacingService Composite

Fig. 19 Simplified DEN-ng contract-service model fragment

operate properly, but is not directly visible to the user. For example, a VPN is visible to the user, but the particular types of forwarding services performed within the VPN are not. A ServicePackage defines the invariant attributes, methods, relationships, and constraints for a collection of CustomerFacingServiceComposite entities. Conceptually, this is a set of related entities that exhibit variations on a set of Services. For example, a ServicePackage could define Platinum, Gold, Silver, Bronze, and BestEffort Services, the difference being the set of applications, their associated QoS and cost, and other factors that are used to differentiate them. A ServiceBundle is a collection of traffic classification and conditioning services that are associated with a particular ServicePackage. Each CustomerFacingService is defined in the Policy Continuum, along with its mapping to the appropriate set of ResourceFacingServices. The Knowledge Continuum is then used to identify the required semantic elements for this mapping.

7 Stability and Uncertainty

If we are to build systems that respond predictably to contextual triggers, we cannot avoid issues of uncertainty and stability. How do these issues manifest themselves within the policy framework?

One way to view a context aware system is that it converts a single *correct* system behavior into a family of *acceptable* behaviors, from which the adaptive system selects *one* behavior to manifest according to the context. In this view there are two complementary notions of correctness [33]:

1. The system behaves correctly in a given context (“point” correctness);
2. The system’s behavior evolves in a way that is consistent with the user’s on-going goals and experience (“process” correctness).

The goal of the adaptive system, then, is to select sequences of point-correct behaviors from the collection of acceptable behaviors, in such a way as to match the adaptations in a process-correct way.

As a concrete example, a policy system managing delivery of a video stream in a wireless network can select from a number of possible bit-rates at any time, depending on the changing capacity of the wireless channel. Point-correctness here means that the video is delivered at *some* acceptable bit-rate. Another policy goal might be to minimize the rate of change of bit-rate over time, so (for example) the picture degrades progressively rather than jumping from high-fidelity to low-fidelity suddenly. This process goal constrains the selection of bit-rates further, with a view to providing a better on-going experience. The challenge for adaptive systems is to ensure that a given set of policies meet these combined goals.

In general we will typically see process goals that seek to minimize sudden changes, essentially seeking to make adaptation a smooth process: small changes in context give rise to small changes in behavior. This not always possible, or desirable – entering a region of radically lower wireless connectivity will necessitate a sudden change in video quality, for example—but one may at least say that such cases highlight situations in which change will be *exceptional*.

Discontinuous adaptations of this kind may seem unimportant. But they interact badly with the uncertainty of sensing and inference. If a system is near such an adaptation point, uncertainty may “push it over the edge” even though the context does not, in reality, require it. Worse, a system may oscillate between two very different behaviors, purely because the decision point lies within the bounds of uncertainty of the sensing system. In such cases an adaptive system may perform more poorly than the simpler systems they were intended to replace.

It is still unclear how such properties can be enforced in policy-based systems. Early work on the semantics of adaptive systems (for example, [38]) has focused on identifying structures in the context that can be used to structure adaptive behavior. Techniques from dynamical systems have also been effective in simple cases [39], allowing closed-form models of adaptive spaces to be derived and analyzed to guarantee the stability and continuity properties of the adaptations. It remains to be seen whether this approach is applicable to policy-based systems, although there seems no reason a priori why this should not be the case.

8 Summary and Future Work

This paper has described a fundamental problem in network management—the inability to relate business concerns to network services and resources provided. Our solution is based on a novel context-aware policy model that, in combination with semantic reasoning facilitated by the use of ontologies, enables us to map business concerns into network services and resources. The use of context ensures that our approach can change the services and resources offered to meet changing business objectives, user needs, and environmental conditions as indicated by changing context.

Initial experience indicates that our approach holds significant promise. However, important challenges remain to be overcome in the areas of semantic mapping; coping with uncertainty; and testing of scalability.

Firstly, policy- and ontology-based management systems will offer significant advantages over more hard-coded approaches, only if issues of stability and robustness to uncertainty can be addressed. These issues require significant additional exploration.

Secondly, current semantic mapping approaches can be characterized as follows: knowledge engineers engage in "one shot" processes resulting in static "one size fits all" mappings published for indiscriminate use. What is needed, however, are semantic mapping approaches that are undertaken across the Policy and Knowledge Continua by a wide range of actors over a period of time, resulting in context-appropriate mappings that need to be tracked, managed, evaluated and evolved.

Finally, in the absence of well-founded approaches, there remains a need to stress-test policy-based systems. We have observed from experience that data sets of sufficient size, quality and annotation to perform adequate testing remain rare, and it would be highly desirable if anonymized data could be made available from production facilities. We plan to address these issues through further collaborative research and through collaboration with others within the Autonomic Communication Forum (www.autonomic-communication-forum.org/).

References

1. Strassner, J.: Autonomic Networks and Systems: Theory and Practice. NOMS Tutorial, April 2008, Brasil (2008)
2. Choi, M., Won-Ki Hong, J.: Towards management of next generation networks. IEICE. Trans. Commun. **E90-B**(11), 3004–3014 (2007)
3. Mo Li, Sandrasegaran, K.: Network management challenges for next generation networks. IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05), pp. 593–598
4. Clark, D., Sollins, K., Wroclawski, J., Katabi, D., Kulik, J., Yang, X., Braden, R., Faber, T., Falk, A., Pingali, V., Handley, M., Chiappa, N.: Newarch: future generation internet architecture. Final technical report (2003)
5. Strassner, J., Menich, B., Johnson, W.: Providing seamless mobility in wireless networks using autonomic mechanisms. ACM Conference on Autonomic Infrastructure, Management, and Security (AIMS), Oslo, Norway, 21–22 June 2007
6. Mitola, J.: Cognitive radio. Ph.D. thesis, KTH, Stockholm, Sweden (2000)
7. Strassner, J.: Policy Based Network Management. Morgan Kaufman, ISBN 1-55860-859-1

8. Dey, A.: Providing architectural support for building context-aware applications. Ph.D. Thesis (2000)
9. Gu, T., Wang, X., Pung, H., Zhang, D.: An ontology-based context model in intelligent environments. Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, vol. 2004 (2004)
10. Keeney, J., Lewis, D., O'Sullivan, D.: Ontological semantics for distributing contextual knowledge in highly distributed autonomic systems. J. Netw. Syst. Manag. **15**, 75–86 (2007)
11. Coyle, L., Neely, S., Stevenson, G., Sullivan, M., Dobson, S., Nixon, P.: Sensor fusion-based middleware for smart homes. Int. J. Assist. Robotics Mechatron. **8**, 53–60 (2007)
12. Strassner, J., Menich, B.: Fusion of sensory information, internal models, and policy in autonomic computing systems. In: Modeling decisions for artificial intelligence (MDAI), Terragona, Spain, 3–5 April 2006
13. Fowler, M.: Dealing with roles. http://www.martinfowler.com/apsupp/roles.pdf (1997). Accessed 28 Sept 2008
14. Bäumer, D., Riehle, D., Siberski, W., Wulf, M.: The role object pattern. http://st-www.cs.uiuc.edu/users/hanmer/PLoP-97/Proceedings/riehle.pdf. Accessed 28 Sept 2008
15. Strassner, J., Agoulmine, N., Lehtihet, E.: FOCALE—a novel autonomic networking architecture. Int. Trans. Syst. Sci. Appl. (ITSSA) J. **3**, 64–79 (2007). ISSN 1751-1461
16. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Trans. Auton. Adapt. Syst. **1**(2), 223–259 (2006)
17. Damianou, N., Bandara, A., Sloman, M., Lupu, E.C.: A Survey of Policy Specification Approaches. Department of Computing, Imperial College of Science Technology and Medicine, London (2002)
18. Damianou, N., Dulay, N., Lupu, E.C., Sloman, M.: The ponder policy specification language. LNCS Proceedings, IEEE 2nd International Workshop on Policies for Distributed Systems and Networks, pp. 18–38 (2001)
19. Kagal, L., Finin, T., Joshi, A.: A policy language for pervasive systems. Fourth IEEE International Workshop on Policies for Distributed Systems and Networks, Lake Como, 4–6 June 2003
20. Román, M., et al.: A middleware infrastructure for active spaces. IEEE Pervasive Comput. **1**(4), 74–83 (2002)
21. Khedr, M., Karmouch, A.: Negotiating context information in context-aware systems. IEEE Special Issue Context Aware Appl. **19**(6), 21–29 (2004)
22. Ye, J., Coyle, L., Dobson, S., Nixon, P.: Ontology-based models in pervasive computing systems. Knowl. Eng. Rev. **22**(4), 315–347 (2007)
23. van der Meer, S., Davy, S., Davy, A., Carroll, R., Jennings, B., Strassner, J.: Autonomic networking: prototype implementation of the Policy Continuum. In: Proceedings of 1st IEEE International Workshop on Broadband Convergence Networks (BcN 2006), pp. 1–10. IEEE, New York
24. Davy, S., Jennings, B., Strassner, J.: The Policy Continuum—a formal model. In: Jennings B., Serrat J., Strassner J. (eds.) Proceedings of the 2nd International IEEE Workshop on Modelling Autonomic Communications Environments (MACE), Multlicon Lecture Notes No. 6, pp. 65–78. Multicon, Berlin (2007)
25. Strassner, J.: Enabling autonomic network management decisions using a novel semantic representation and reasoning approach. Ph.D. thesis (2008)
26. Posnak, E., Lavender, R.G., Vin, H.: Adaptive pipeline: an object structural pattern for adaptive applications. 3rd Pattern Languages of Programming conference (1996). Accessed 8 March 2008
27. Festinger, L.: A Theory of Cognitive Dissonance. Stanford University, Stanford University Press, Stanford (1957)
28. Wong, A., Ray, P., Parameswaran, N., Strassner, J.: Ontology mapping for the interoperability problem in network management. J. Sel. Areas Commun. **23**(10), 2058–2068 (2005)
29. Liu, Y., Zhang, J., Jiang, M., Raymer, D., Strassner, J.: A case study: a model-based approach to retrofit a network fault management system with self-healing functionality. 15th IEEE International Conference on Engineering of Computer-Based Systems, Belfast, Northern Ireland, 31 March 2008
30. Strassner, J.: DEN-ng model overview. Joint ACF, EMANICS, and AutoI Workshop on Autonomic Management in the Future Internet, 14 May 2008
31. Strassner, J., de Souza, J.N., Raymer, D., Samudrala, S., Davy, S., Barrett, K.: The design of a novel context-aware policy model to support machine-based learning and reasoning. J. Clust. Comput. **12**(1), 17–43 (2009)
32. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. In: Proceedings of the First International Conference on Pervasive Computing

(Pervasive ‘02), London, UK, vol. 2414, pp. 167–180. ISBN 978-3-540-44060-4 Springer, Berlin/Heidelberg (2002)

33. Coutaz, J., Crowley, J., Dobson, S., Garlan, D.: Context is key. Commun. ACM **48**(3), 49–53 (2005). ISSN 0001-0782
34. Noy, N.: Semantic integration: a survey of ontology-based approaches. SIGMOD Rec. **33**(4), 65–70 (2004)
35. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Berlin (2007). ISBN: 9783540469113
36. O'Sullivan, D., Wade, V., Lewis, D.: Understanding as we roam. IEEE Internet Comput. **11**(2), 26–33 (2007)
37. Keeney, J., Lewis, D., O'Sullivan, D., Roelens, A., Boran, A.: Runtime semantic interoperability for gathering ontology-based network context. 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), pp. 56–65. Vancouver, Canada, April 2006
38. Ye, J., Coyle, L., Dobson, S., Nixon, P.: A unified semantics space model. In: Location- and context-awareness, vol. 4718, pp. 103–120. LNCS 2007
39. Dobson, S., Bailey, E., Knox, S., Shannon, R., Quigley, A.: A first approach to the closed-form specification and analysis of an autonomic control system. In: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems. Auckland, NZ 2007

Author Biographies

John Strassner is the director of autonomic research in the Telecommunications Systems & Software Group in Waterford Institute of Technology, and a Visiting Professor at POSTECH. His research interests are in autonomic systems, policy based management, machine learning, and semantic reasoning. He is the Chairman of the Autonomic Communications Forum, and the past chair of the TMF's NGOSS SID, metamodel and policy working groups. He has authored two books, written chapters for five other books, and co-edited five journals on network and service management and autonomics. John is the recipient of the Daniel A. Stokesbury memorial award for excellence in network management, and has authored 211 refereed journal papers and publications.

Sven van der Meer received his M.Sc in computer science and his Dr.-Ing. from Technical University Berlin (TUB), Germany, in 1996 and 2002. Since November 2002, Sven has been a research fellow at the Telecommunications Software & Systems Group at the Waterford Institute of Technology. Since October 2004 he is Senior Investigator of the Competence Centre for Communication Infrastructure Management at TSSG, involved in the Architecture and Information Modelling teams in the TMF, and has served as editor for Technological Neutral Architecture and Contracts specifications within the TM Forum.

Declan O'Sullivan is the director of the Knowledge and Data Engineering (KDEG) research group in Trinity College Dublin (TCD). His research interests are in the use of semantic-driven approaches for network and service management, in particular to enable semantic interoperability. He is currently a Principal Investigator in the SFI funded research project investigating Federated Autonomic Management Environments (FAME). O'Sullivan has a Ph.D. and a M.Sc in computer science from TCD.

Simon Dobson is a co-founder of the Systems Research Group at UCD Dublin. His research centers around adaptive pervasive computing and novel programming techniques. He is on the editorial boards of the Journal of Network and Systems Management and the International Journal of Autonomous and Adaptive Communications Systems, and participates in a number of EU strategic workshops and working groups. He is National Director and vice-president of the European Research Consortium for Informatics and Mathematics, a board member of the Autonomic Communication Forum, and a member of the IBEC/ICT Ireland standing committee on academic/industrial research and development. He holds a BSc and DPhil in computer science, is a Chartered Fellow of the British Computer Society, a Chartered Engineer, and member of the IEEE and ACM.