The role of ontologies in autonomic computing systems

by L. Stojanovic
J. Schneider
A. Maedche
S. Libischer
R. Studer
Th. Lumpp
A. Abecker
G. Breiter
J. Dinger

The goal of IBM's autonomic computing strategy is to deliver information technology environments with improved self-management capabilities, such as self-healing, self-protection, self-optimization, and self-configuration. Data correlation and inference technologies can be used as core components to build autonomic computing systems. They can also be used to perform automated and continuous analysis of enterprise-wide event data based upon user-defined configurable rules, such as those intended for detecting threats or system failures. Furthermore, they may trigger corrective actions for protecting or healing the system. In this paper, we discuss the use of ontologies as a high-level, expressive, conceptual modeling approach for describing the knowledge upon which the processing of a correlation engine is based. The introduction of explicit models of state-based information technology resources into the correlation technology approach allows the construction of autonomic computing systems that are capable of dealing with policy-based goals on a higher abstraction level. We demonstrate some of the benefits of this approach by applying it to a particular IBM implementation, the eAutomation correlation engine.

The increasing complexity of information technology (IT) systems demands a correspondingly greater effort for systems management. Today, many systems management tasks such as system configuration, performance analysis, performance tuning, error handling, and availability management are often performed manually. This work can be time-consuming and error-prone. Moreover, it requires a growing number of highly skilled personnel, making IT systems costly. IBM's autonomic computing initiative,[1] which is a core element of IBM's e-business on demand* strategy,[2] addresses this problem by developing and providing powerful concepts for self-management, including new self-healing, self-protecting, self-optimizing, and self-configuring capabilities. The goal is to reduce the burden associated with the management and the operation of IT systems. Autonomic computing systems should simply work, repairing and tuning themselves as needed. This requires that such systems be able to protect themselves, to identify upcoming problems, and to make required reconfigurations dynamically in order to resolve problems.

As one important step toward a vision of autonomic computing systems, IBM has developed several correlation engines.[3,4] These include ABLE (Agent Building and Learning Environment), AMIT (Active Middleware Technology), eAutomation, TEC (Tivoli Enterprise Console*), Yemanja, and ZCE (Zurich Correlation Engine). Correlation engines are autonomic core components that perform continuous automated analysis of enterprise-wide, normalized, real-time event data based on user-defined configu-

0018-8670/04/$5.00 © 2004 IBM

rable rules. These rules can be used to detect threats, complex attack patterns, or system failures, and to initiate a corresponding reaction.

Today IBM's correlation engines are being used successfully in many application domains. Examples include the eAutomation correlation engine in System Automation for OS/390* [5] and System Automation for Multiplatforms,[6] and the Zurich Correlation Engine for router fault isolation and for intelligent monitoring capabilities.

In this paper we propose the application of formal ontologies[7,8] as the conceptual backbone for correlation engines. In philosophy an ontology is a theory about the nature of existence and, in particular, about what types of things can exist; ontology as a discipline studies such theories. Artificial intelligence and Web researchers have adopted this term for their own purposes. For them an ontology describes a formal, shared conceptualization of a particular domain of interest.[7] Thus, ontologies provide a way of capturing a shared understanding of a domain that can be used both by humans and systems to aid in information exchange and integration. The use of ontologies has several advantages.

First, ontologies can facilitate interoperability between correlation engines by providing a shared understanding of the domain in question. In this way problems caused by structural and semantic heterogeneity of different models can be avoided. Structural heterogeneity results when different correlation engines store their data in different schemes. Semantic heterogeneity involves similar problems in the content and intended meaning of information. Ontologies provide an effective means for explicating implicit design decisions and underlying assumptions at system build time. This makes it easier to reason about the intended meaning of the information interchanged between two systems. Hence, interoperability is a key benefit of the application of ontologies, and many ontology-based approaches to information integration have been developed.[9]

Second, ontologies provide a formalization of shared understanding which allows machine processability. Machine processability in turn forms the basis for the next generation of the World Wide Web, the so-called Semantic Web,[10,11] which is itself based on using ontologies for enhancing (annotating) content with formal semantics. This will enable autonomic agents to reason about Web content and to carry out more intelligent tasks on behalf of the user.

Finally, the explicit representation of the semantics of data through ontologies will enable correlation engines to provide a qualitatively new level of services, such as verification, justification, and gap analysis, as we discuss later in this paper. These engines will be able to weave together a large network of human knowledge and will complement this capability with machine processability. Various automated services will then aid users in achieving their goals by accessing and providing information in a machine-understandable form. It is important to note that ontologies not only define information, but also add expressiveness and reasoning capabilities. Ontology rules provide a way to define behavior in relation to a system model.

The focus of this paper is on the third, most ambitious benefit that can be achieved by using ontologies in autonomic computing, namely the providing of new levels of services. To illustrate this benefit we will use the eAutomation correlation engine technology and its resource relationship model.

This paper is organized as follows: the next section establishes the terminology of correlation technology through the definition of a reference model. We then discuss the basic concepts of the Semantic Web. In the following section we combine these two different technologies by using an ontology as a model for the eAutomation engine. The practicability of this approach is emphasized as we then describe the benefits of the ontological approach. After a discussion of related efforts, we conclude by outlining directions for future work.

A reference model for correlation engines

This section defines the basic concepts required for describing and reasoning about correlation technology. We then introduce a correlation-engine reference model, which provides a layered structure of corresponding elements. It also allows identification of technology building blocks and of the different correlation patterns required for the design of autonomic systems.

Autonomic computing systems constantly monitor and gather the data they need to react to or act upon, according to their management tasks and targets. This data is elaborated and organized through the notion of events, which we define more formally in a later section. Events in turn are typically meaningful in a certain context when correlated with other

events. Correlation technology can then be used for the following types of analyses:

- Data filtering
  Example: The system detects a network-down failure but suppresses subsequent redundant notifications regarding this particular problem.
- Thresholding
  Example: If the restart of an application fails more than twice, the application is considered to have a severe problem.
- Sequencing
  Example: If a door to a secure area opens and does not close within a certain time interval, an alert is sent.

The output of such analyses can be newly created events describing the occurrence of a potentially interesting (for example, dangerous, valuable, or important) situation that has been identified. This new event can then be fed to other correlation rules as input. An additional layer of correlation rules is required to trigger concluding actions based both on the input and on other domain-specific data. For example, events can be related to resource state changes, and such state changes may in turn be correlated to other related resources and their respective current states. The eAutomation correlation engine described later is an example of such an autonomic computing system.

In general, autonomic computing systems based upon correlation technologies are able to:

1. React to a problem that happens in the defined problem space (e.g., if a server fails, direct requests to other servers).
2. Use predictive methods to discover potential problems in order to achieve better results and eliminate problems (e.g., if utilization of a particular server is high, direct requests to other servers).

These tasks require a deep knowledge of the problem space (i.e., the scope of possible problems) of a particular systems management discipline, such as availability or performance, because the problem can be resolved systematically only by analyzing and exploiting the interdependencies among components or IT resources.

Correlation engines can be used as key components to build a MAPE (Monitor, Analyze, Plan, Execute) model,[1] which breaks management architecture down into four common functions: (1) collecting data, (2) analyzing data, (3) creating a plan of action, and (4) executing the plan. By monitoring (M) behavior and analyzing (A) data, planning (P) the actions that should be taken, and executing (E) them, a kind of a control loop is created.

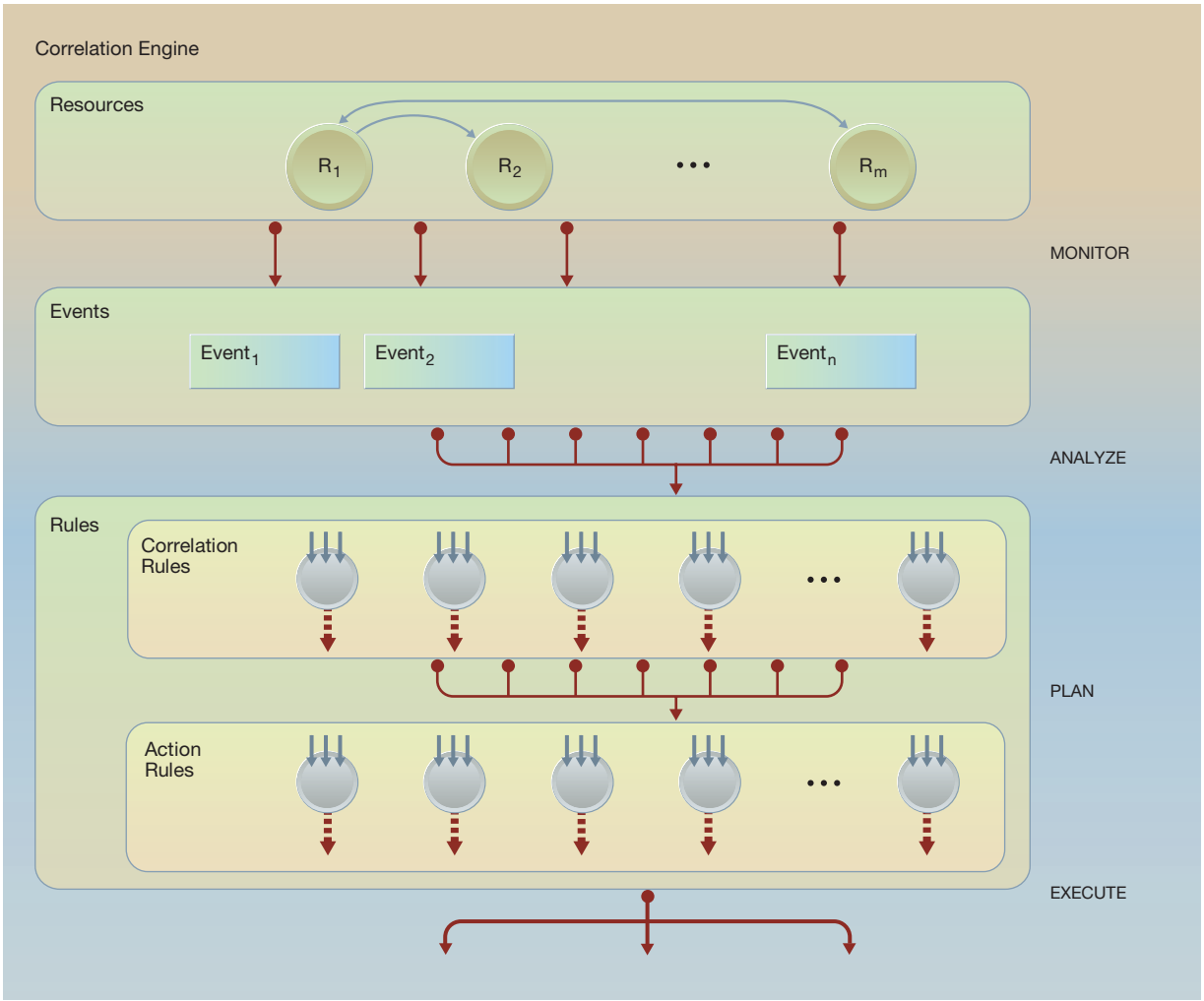
The MAPE model assumes the existence of a common knowledge element[1] that represents the knowledge about a problem space that is shared among the four components of the MAPE model. This shared knowledge includes such aspects as topology information, system logs, performance metrics, and policies that are relevant to the problems that can be resolved with a correlation engine. However, the MAPE model does not provide guidelines regarding how to represent the knowledge about a certain management discipline, how to acquire data about problems in a domain, and how to use that information for resolving problems without the need for a high level of human intervention. Consequently, each correlation engine has its own knowledge model, as illustrated in Reference 12. To solve the resulting problems of data integration, we have defined a general reference model for correlation engines by abstracting and describing knowledge hidden in the knowledge element on the conceptual level. This model is shown in Figure 1. It is obtained from the MAPE model by considering the data processed by a correlation engine rather than the steps in that processing.

The reference model consists of three layers:

1. The resource layer
2. The event layer
3. The rule layer

A resource can be almost anything, including both real-world entities, such as a specific piece of hardware (a CPU) or software (a database), and virtual entities, such as business applications or logical IT services. The resource model is central to representing what is being managed. An event represents a significant change in the state of a resource (e.g. failure of a CPU), and it is generated to provide notification about resource changes or problems. An automated response to an event is typically triggered by either correlation rules or action rules. Correlation rules try to discover a problem, for example, by translating several dependent events into one meaningful event. Action rules trigger actions that try to resolve a problem, for example, by executing dedicated programs.

Figure 1 Reference model for correlation engines



The layered reference model for correlation engines enables the separation of:

- What has to be managed
- Why management is required (monitoring)
- How to manage (decision logic)

Consequently, correlation engines are able to:

- Capture and select important events and update key characteristics (states) of the managed resources
- Detect changes or problems based on knowledge of state changes
- Initiate actions to correct any behavior not in line with a desired goal

In the rest of this section, we first describe the resources that have to be managed. We then provide a classification of the events that can trigger the management process. Finally, we describe the rules that represent automated responses to events.

Resources. Unless each resource in a system can share information with every other part and contribute to some overall system awareness, the goal of autonomic computing will not really be reached.

Thus, a common data model is needed that can be used to represent all resources.

Currently the most frequently used data models apply the basic structuring and conceptualization techniques of the object-oriented paradigm.[13] The following characteristics are key components of such models:

- The object constitutes the central modeling construct and carries an object identity.
- Types can be specified for objects, with all objects of one type having the same structure.
- The state of an object is defined by the values of a set of properties. These properties can be either attributes of the object or relationships of the object to other objects.

The possibility of organizing resources into a hierarchy is very important. This accelerates model development and increases the readability and maintainability of a model because inheritance allows the construction of models that are compact and without redundancy. Inheritance is particularly valuable for modeling domains that contain numerous types of similar resources, such as computers, internetworking devices, and databases. It allows management applications to treat resources generically, ignoring their specific details when they are not relevant to the problem at hand. For example, the resource Cisco 827-4V router is a specialized instance of the resource Cisco 800 series router. It inherits all the properties of the Cisco 800 series router. Moreover, it can add new properties or modify the inherited ones.

The capability of expressing resource dependencies explicitly is of paramount importance because a correlation engine involves multiple decisions that entail interactions among resources. Thus, the resource model has to emphasize relationship properties that capture the dependencies among managed resources. Relationship properties represent such information as the fact that a TCP (Transmission Control Protocol) connection is layered over a particular IP (Internet Protocol) link, that a client is using the services of a particular server, or that an application ran on a particular computer. The knowledge of such relationships is essential for almost all autonomic management functions. Furthermore, it is not sufficient just to establish the relationships among resources. The semantics of these relations are also important because without knowing the explicit meaning of a particular type of relationship, resolution of a problem is not possible.

Events. An event is a special kind of a message generated by a resource in the domain that indicates a change of state of that resource. For example, in a managed network, incidents such as component failures, congestion, or a network element reverting to a backup system may cause a change of state that generates an event.

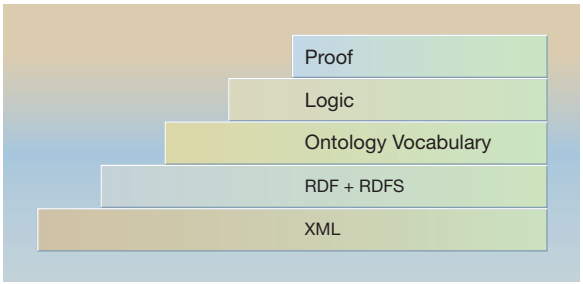
Analyzing event data is difficult if the data is not normalized into a common, complete, and consistent model. This entails not only reformatting the data for better processing and readability, but also breaking it down into its most granular pieces. It includes filtering out unwanted information to reduce analytical errors or misrepresentations. It can even involve acquiring more information from outside the scope of the original event data, for example, from the operating system on which an event source is running.

Moreover, different management disciplines require different types of events. In telecommunications systems events are described in International Telecommunications Union (ITU) standards documents, but are different from one network type and protocol to the next. Therein lie the challenges for the model designer—to allow virtually any type of event to be defined and to provide maximum infrastructure for supporting event handling.

Indeed, the prerequisite for meaningful analysis of an event is that information about that event be properly organized and interpreted. Thus, it is important to express information about events in a common and uniform way, which implies in turn that a model of events is necessary. The role of such a model is to describe what happened, why it happened, when it happened, and what the cause was. Additional support information for decision making may also be incorporated, such as cost analysis, prioritization, and asset allocation information. This additional data enables a thorough analysis, but, in order to separate the monitoring logic and the decision logic, this model must not contain information about how the event should be resolved.

Rules. As noted previously, rules can be divided into two categories based upon their roles in autonomic computing systems: (1) correlation rules and (2) action rules. We describe these types of rules in the next two sections.

Figure 2 Layers of the Semantic Web architecture



Correlation rules. One task of the correlation engine is to reduce the number of events shown, for example, to the system administrator, and to enrich the meaning of the events. Ideally, the correlation engine should be able to condense the received events into a single event directly indicating a problem (i.e., a situation event) in the managed system. For example, a rule might specify that the administrator is to be notified only if three memory problems occur within an hour.

Correlation rules can be divided in two types:

- Stateless rules consider events in isolation. They perform passive filtering on the attribute values of an incoming event. For example, a specific stateless rule detects a system failure when a file system has crashed or an IP address has failed.
- State-based rules are critical for analyzing events over time. They allow the same or repeating events to be distilled into a single event, regardless of the frequency of occurrence. For example, a rule might require that the administrator be alerted if an IP address is involved in five separate attacks on different parts of the network over a 6-month period.

Thus, stateless correlation rules operate on a single current event, whereas state-based correlation rules rely on a history of events.

Action rules. Preprocessing, filtering, and correlating events before they are passed to the next level of autonomic manager or directly to the operating staff minimizes the time spent on repairs, provides more specific alarm information, and clarifies fault correlation. However, in order to automate corrective actions, additional inference rules and designated action rules might be needed. These rules are used to reduce a system administrator's work in two ways:

- By triggering automatic remedy actions
- By gathering additional monitoring data to obtain a detailed view of the current exceptional state of resources; this additional information should reduce the efforts a system administrator must make to decide how to treat an affected resource. For example, in the case of a printer jam, action rules can be used to restart the printer or to inform the administrator about the printer failure.

The Semantic Web

The main goal of the Semantic Web is to be able to express the meaning of resources that can be found on the Web.[11] In order to achieve that objective, several layers of representational structures are needed.[10] The subset of these layers that is relevant for our discussion is presented in Figure 2.

These layers have the following roles:

- The XML (eXtensible Markup Language)[14] layer represents the structure of data.
- The RDF (Resource Definition Framework)[15] layer represents the meaning of data.
- The ontology layer represents the formal common agreement about the meaning of data.
- The logic layer enables intelligent reasoning with meaningful data.
- The proof layer supports the exchange of proofs in an interagent communication, enabling common understanding of how the desired information is derived.

It is worth noting that the real power of the Semantic Web will be realized when many systems are created that (1) collect Web content from diverse sources, (2) integrate and process the information, and (3) exchange the results with other human or machine agents. Thus, the effectiveness of the Semantic Web will increase drastically as more machine-readable Web content and more automated services become available. This level of interagent communication will require the exchange of proofs to ensure common understanding among these agents.

Two important technologies for developing the Semantic Web are already in place, namely XML and RDF. XML lets users create their own tags to annotate Web pages or sections of text on a page. Systems can make use of these tags in sophisticated ways, but to do so a systems programmer must know what the page author intended by each new tag. In other

words, XML allows users to add arbitrary structure to their documents, but says nothing about what the structures mean.[16] Typically the meaning of XML documents is intuitively clear to humans because the semantic markup and tags use terms that are common in the particular domain. However, computers do not have intuition. Tag names do not in and of themselves provide semantics.

Document type definitions (DTDs) are a possible approach to providing structure for the content of the documents. However, structure and semantics are not always aligned; they can be orthogonal. Therefore, a DTD is not an appropriate formalism to describe the semantics of an XML document. The same holds for XML Schema,[17] which also only define structure, although with a richer language. In essence, XML lacks a semantic model. It has only a surface model, or tree. Thus, XML is not the solution for propagating semantics throughout the Semantic Web. It can only play the role of a transport mechanism as a readily machine-processable data format.

RDF[17] provides a means for adding semantics to a document. RDF is an infrastructure that enables encoding, exchange, and reuse of structured meta-data. Principally information is stored in the form of RDF statements, which are machine-understandable. Search engines, intelligent agents, information brokers, browsers, and human users can understand and use that semantic information. RDF is implementation independent and may be serialized in XML (i.e., its syntax is defined in XML). Adding semantic information to Web documents is called semantic annotation.[18] RDF in combination with RDFS (Resource Description Framework Schema)[19] offers modeling primitives that can be extended to meet the needs of a specific situation. Basic class hierarchies and relations between classes and objects are expressible in RDFS. In general, however, RDFS suffers from a lack of formal semantics for its modeling primitives, making proper interpretation an error-prone process.

A solution to this problem is provided by the third basic component of the Semantic Web, namely ontologies. Ontologies are well-suited for describing the heterogeneous, distributed, semistructured information sources (e.g., XML documents) that can be found on the Web or on intranets. By defining shared and common domain theories, ontologies help both people and machines to communicate concisely by supporting the exchange of semantics rather than just syntax. It is therefore important that any semantics for the Web be based upon an explicitly specified ontology. In this way consumer and producer agents can reach a shared understanding by exploiting ontologies that provide the vocabulary needed for negotiation.

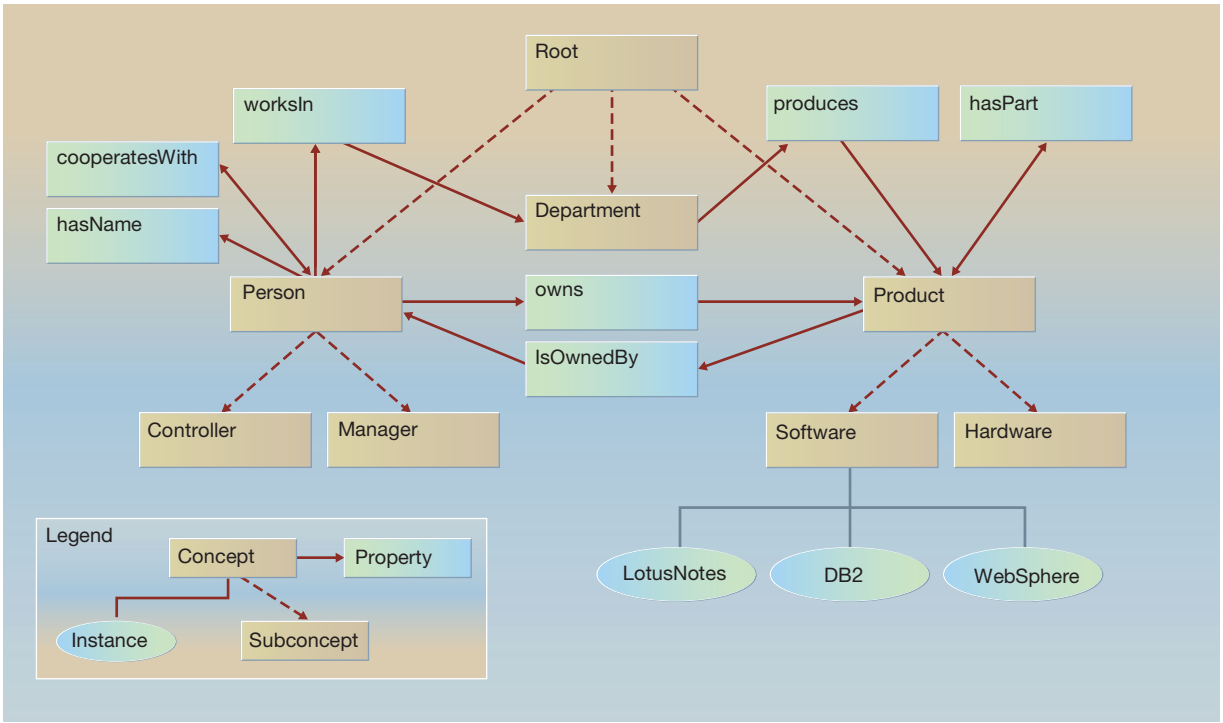
KAON ontologies

Many representation languages for ontologies have been defined, including OIL (Ontology Interchange Language),[20] DAML (DARPA Agent Markup Language) + OIL,[21] and OWL (Ontology Web Language).[22] Our work is based upon the KAON (KArlsruhe ONtology) ontology language definition.[23] Briefly, the KAON ontology language is based on RDFS, but provides a clean separation of the modeling primitives from the ontology itself. KAON provides means for modeling metaclasses and incorporating several commonly used modeling primitives, such as transitive, symmetric, and inverse properties, as well as cardinalities. Figure 3 shows a very simple KAON ontology.

An ontology in the KAON language consists of concepts (sets of elements) and properties (specifications of how objects may be connected).

An ontology in the KAON language consists of concepts (sets of elements) and properties (specifications of how objects may be connected). For example, the ontology shown in Figure 3 contains, among others, the concepts Person, Department, and Product, and the properties worksIn, and produces. Each property must have at least one domain concept; for example, the domain concept for the property worksIn is the concept Person. Its range may either be a literal (e.g., the hasName property), or a set of at least one concept; thus the range concept for the property worksIn is the concept Department. Domain and range concept restrictions are treated conjunctively. Consequently, all of them must be fulfilled for each property instantiation. Some properties may be marked as symmetric (cooperatesWith) or transitive (hasPart). Further, it is possible to say that two properties are the inverse of one another, such as the properties owns and isOwnedBy. Moreover, it is possible to specify rules such as, "If a person works in

Figure 3 Example of KAON ontology



a department, he or she has knowledge about a product produced in that department."

For each concept-property pair it is possible to specify the minimum and maximum cardinalities, defining how many times a property may be specified for instances of that class. Concepts and properties can be arranged in a hierarchy. For example, the concepts Software and Hardware are subconcepts of the concept Product. The hierarchy relationship interrelates directly connected concepts (properties) and is defined as a transitive relationship.

Each ontology has an instance pool associated with it. An instance pool is constructed by specifying instances of different concepts and by establishing property instantiations between instances. Property instantiations must follow the domain and range constraints and must obey the cardinality constraints, as specified by the property specifications.

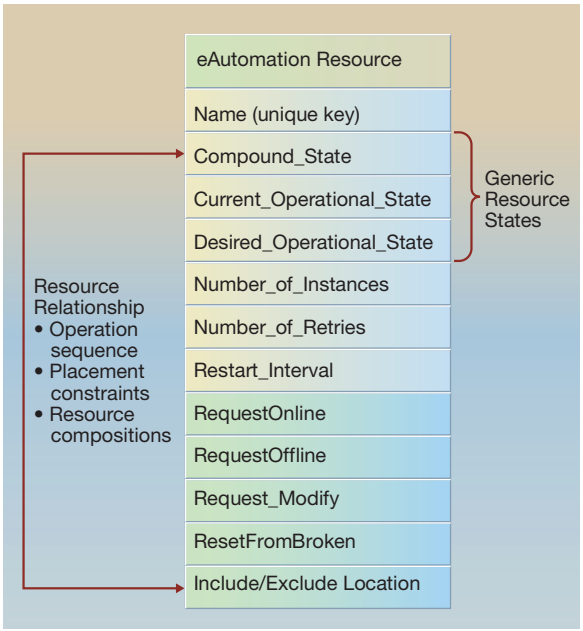
An ontology also contains so-called lexical entries, such as labels, synonyms, stems, or textual documentation, that reflect various lexical properties of ontology entities. There is an m:n relationship between lexical entries and ontology entities. Thus, the same lexical entry may be associated with several elements (e.g. the label APPLE may be associated with an instance representing an apple fruit or an Apple computer).

All information is organized in so-called OI models (ontology-instance models), containing both ontology entities (concepts and properties) and their instances. This allows grouping of concepts with their instances into self-contained units. For example, the ontology modeling IBM products shown in Figure 3 contains the concept Software along with instances representing several well-known products, including DB2*, Lotus Notes*, and WebSphere*. An OI model may include another OI model, thus making all definitions from the included OI model automatically available.

Ontology-based correlation engines

The major objective of this paper is to show how to bring one aspect of autonomic computing, namely the use of correlation engines, to its full potential

Figure 4 Abstract resource model



by extending and combining it with the Semantic Web technologies described previously.

Specifically, we will now focus on how ontologies may advance autonomic-computing solutions. We use the eAutomation correlation engine[24] as an example throughout this section. Note that similar strategies can be applied for other engines as well. The approach can be summarized in the following steps:

1. The model of the eAutomation engine is first transformed into the eAutomation ontology.
2. Hidden (hard-coded) knowledge embedded in the eAutomation engine is translated into a set of rules in the corresponding ontology and is used in typical inferencing tasks.

In the rest of this section we describe the model of the eAutomation engine and show how it can be improved by translating the model of the eAutomation engine into the eAutomation ontology.

The eAutomation engine. Any correlation engine has to provide answers to three questions: (1) what to manage, (2) why management is required, and (3) how to manage. Hence, for the rest of this subsection we elaborate the resource, event, and rule models used in the eAutomation engine.

Resources. One of the core foundations of the eAutomation engine is the abstract representation of any type of IT resource, with the ultimate goal of availability management. This abstract resource representation is depicted in Figure 4. (Note that we will not discuss all elements of the abstract resource model explicitly in this paper.) Each resource has a unique name that is represented through the Name attribute. The most important attributes of a resource are those related to the operational state of a resource. Whereas the Current_Operational_State attribute describes the current availability state of a resource, which is typically monitored and whose change is represented as an event, the Desired_Operational_State attribute is used to specify the desired state of a resource. This state is typically set by an administrator or is part of a policy reflecting overall goals. The Compound_State attribute indicates the state of a resource in the context of other resources together with the composition of the states of these other resources, thus providing a view of the overall situation. This state is computed based on internal aggregation; a correlation rule is used to derive an overall state. The set of possible values of the state attributes is predefined. For example, if the resource is not started, the value of the Current_Operational_State attribute is Offline. The Online value means that the resource is ready for work, whereas Pending Online means that the resource has been started, but is not yet ready for work. There are several operational states indicating problems. For example, Failed Offline signifies that a resource is broken and cannot be used.

The eAutomation resource model allows resources to be grouped in resource groups, a process known as composition. A resource group is itself an eAutomation resource which contains a collection of resources that are handled as one logical entity. Different types of groups can be supported by implementing different state aggregation rules. These rules are used to derive the observed state of the group resource. Because a resource group is just another resource, it can in turn be a member of another resource group. Internally, we build an isMemberOf relationship tree.

An equivalency is a collection of resources that provide the same functionality. An equivalency consists of a set of resources from the same class. For example, network adapters might be defined as members of an equivalency. If one network adapter fails,

another network adapter can take over processing. In essence, an equivalency is a resource group, but it does not have its own state.

The eAutomation correlation engine allows the definition of two types of relationships between resources:

1. *Start/stop relationships*—These are used to define the start and stop dependencies between resources. A startAfter relationship indicates that resource X will only be started after resource Y has been started. This is an active relationship because when there is a start request (i.e., the Desired_Operational_State attribute is set to Online) for X, eAutomation will propagate this requirement to Y along the relationship subgraph. Start/stop relationships can provide even more complex behavior, which can be described by the function *dependency.* A dependsOn relationship is used when a resource X cannot operate properly if another resource is offline. A dependsOn relationship actually implies a startAfter requirement.
2. *Location relationships*—These are used for locating resources on nodes or other hosting containers. Several relationships are defined: Collocated, AntiCollocated, Affinity, AntiAffinity, and IsStartable. The Collocated relationship requires that a resource A can only be started at a location where a resource B is already running. AntiCollated means that resource A must be started at a location different from that where resource B is running. These are hard location relationships, meaning that the location relationship must be fulfilled. The Affinity and AntiAffinity relationships have a similar behavior, except that these are soft location relationships that can be ignored in problem situations. An example is an AntiAffinity relationship between the resources A and B when only one node in a cluster is available. Here, both resources are allowed to run on the same node. The IsStartable relationship between A and B defines that a resource A can only be placed on a node where a resource B is startable.

Events. On the level of the eAutomation engine all events are related to changes of the states of resources. The event correlation techniques described earlier can be used by agents to derive these states.

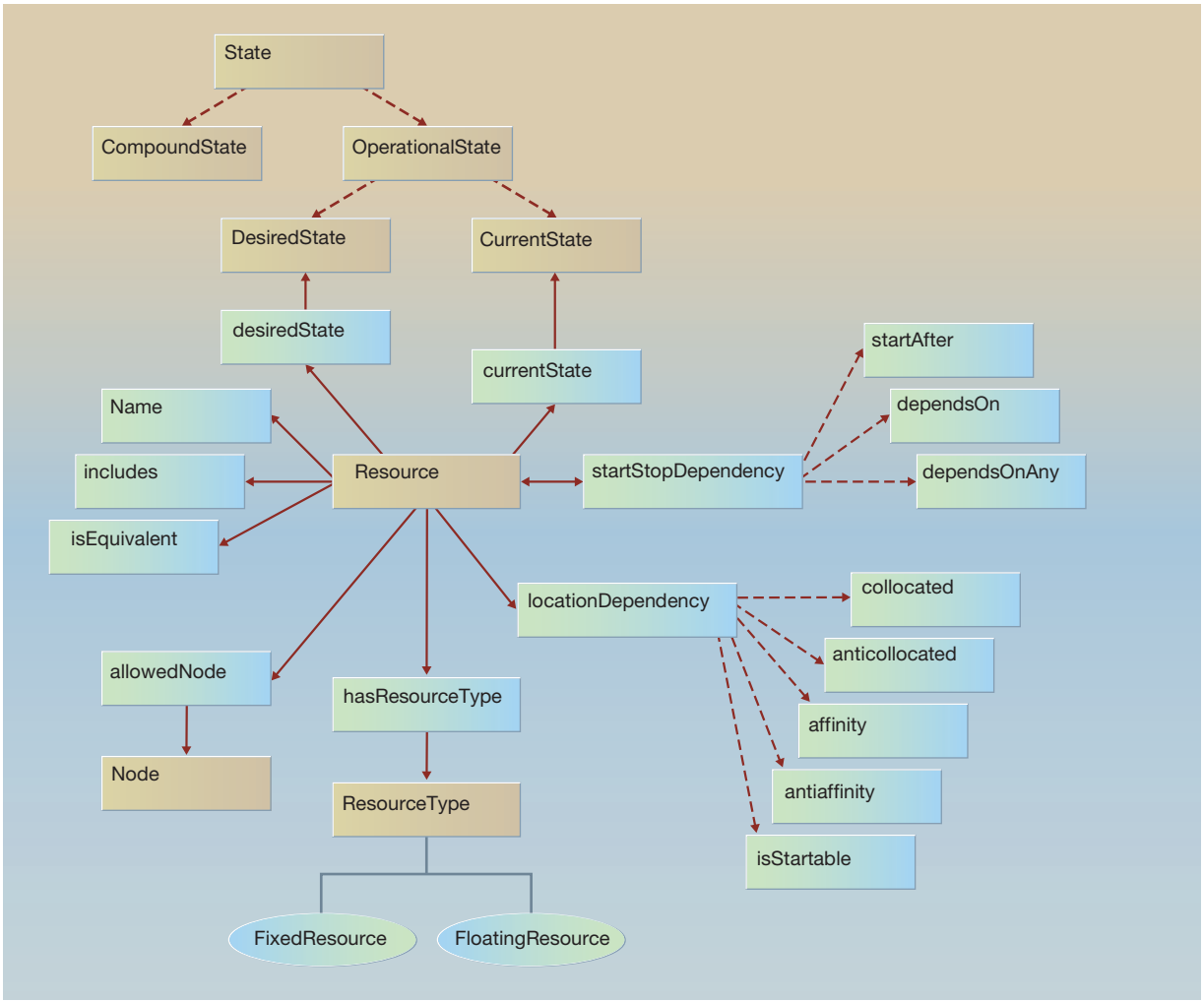
Rules. The eAutomation correlation engine combines correlation and action rules into the following three types of rules:

1. *Simple correlation rules*—These rules take the form
   WHEN condition
   THEN action
   The condition is an expression of resource attributes. The evaluation takes place when one of these attributes changes. An action can be, for example, the setting of an attribute value.
2. *Relationship correlation rules*—This type of rule is based upon the dependencies between resources described in a model. These rules consider attributes of related resources following a relationship subgraph. In particular they describe the relationship subgraph that must be traversed and the specific attributes that must be considered in a given situation. An example of a relationship correlation rule is one that states that when all predecessor resources related through the startAfter relationship are Online, then a successor resource can be started. This rule will be evaluated whenever there is a state change in one of the resources involved.
3. *Request propagation rules*—Requests can be propagated along a subgraph, and they can be transformed during that propagation. For example, when there is a start request for resource A, this request is propagated down to all components with which A has a startAfter relationship, because all these other resources must be up and running before A can start. The propagation stops at resources that are at the end of startAfter relationship chains. At these so-called leaf resources, start commands are sent out as decisions.

Although simple correlation rules combine stateless rules (i.e., rules that perform only filtering), we could argue that making decisions based on state propagation rules and complex relationship rules extends into the regime of inferencing technology.

It should be noted that none of the inference rules described above are externalized to the customer user interface. A customer defines an eAutomation policy based on its IT resources and the desired automation behavior described through resource groups, equivalencies, and relationships. This information is then internally translated into resource model inference rules for the system.

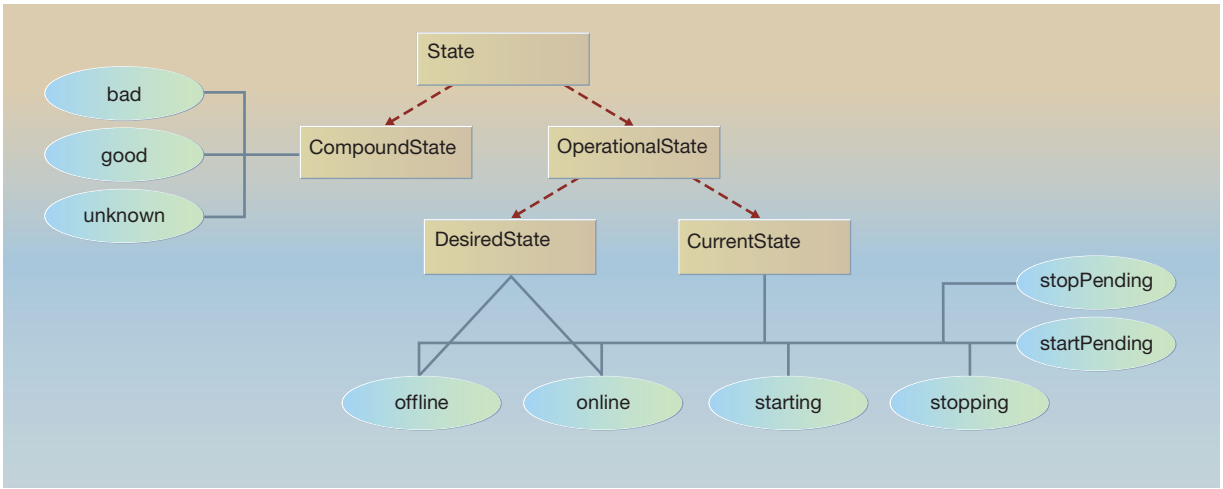
Figure 5 Part of the eAutomation ontology



The eAutomation ontology. The model of the eAutomation engine presented in the previous section was then used as the basis for defining the eAutomation ontology. Our goal was to model all information that exists in the eAutomation model, including any implicit knowledge. Here we briefly describe the process of enriching the eAutomation model into the eAutomation ontology.

Figure 5 shows a part of the eAutomation ontology. The most important concept in this ontology is the concept Resource, corresponding to the abstract resource in the eAutomation model shown in Figure 4. Each ontology concept is described with a set of properties that can be either attributes or relations. The concept Resource contains only the one attribute name; the value for this property can be an arbitrary string (e.g., Resource123). All other properties defined for the concept Resource are relationships. For example, the current state of a resource is modeled as the property currentState (the range of this property is the concept CurrentState) because the set of possible values is known in advance. In this way the correctness of the concrete model is improved significantly because only instances of the concept CurrentState can be used to specify the value of the current resource state. Note that domains and ranges simply specify schema constraints that must be sat-

Figure 6 The concept "state" in the eAutomation ontology



isfied for the property to be instantiated. They do not infer new facts, but rather guide the user in constructing the ontology by determining what can and cannot be explicitly stated.

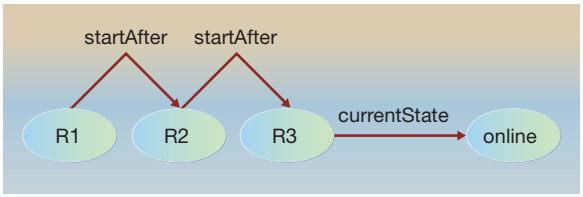
In order to define the state of a resource in a unique way, the concept State is organized into the concept hierarchy shown in Figure 6. The concept State contains two subconcepts. The first subconcept CompoundState has the instances good, bad, and unknown. These instances represent the concrete values that can be used for the property instances. Note that good is a unique identifier for an instance of the concept CompoundState and is much more than just the string "good." When good is considered as a URI (unique resource indicator), additional values can be specified according to the structure of the concept CompoundState. Further, lexical entries can be defined. For example, the instance good can be related to the term (string) "gut" in German. The second subconcept OperationalState is divided into the concepts DesiredState and CurrentState. The possible values for the operational state are modeled as instances. The values that are specific only for the current state (e.g., stopping) are represented as instances of that concept.

Several properties of resources are derived from the model of the eAutomation correlation engine. These include startAfter, dependsOn, dependsOnAll, collocated, anticollocated, affinity, antiaffinity, and isStartable among others. Based on their semantics, these properties are organized into two groups: startStopDependencies (used to define a start/stop behavior) and locationDependencies (used for locating resources on nodes). These groups correspond to the relationships that can be defined between resources in the eAutomation model.

Each of these relationships has an implicit meaning that is hard-coded in the program that uses them. Moreover, there are semantic connections between some of these relationships. For example, the collocated relationship is symmetric; the collocated and anticollocated relationships are mutually inverse. This knowledge may be defined formally and explicitly by axioms.

In an ontology there are two types of implicit knowledge: axioms and general rules. Axioms are a standard set of rules, such as the rules for symmetric, transitive, and inverse properties. For example, if A contains B, B contains C, and contains is a transitive property, then the ontology system can infer that A contains C as well. Thus, we do not need to express this information explicitly. General rules are domain-specific rules that are needed to combine and to adapt information available in the ontology. They are used to specify the relationships between ontological entities in the form of rules. For example, if A contains B and B is about C, then it can be concluded that A also is about C.

Figure 7 Inferencing in KAON



In general, axioms and rules are used to infer new knowledge. The possibility of deriving information makes the model of a domain more concise, more accurate, and easier to maintain. Obtaining and formalizing the nonexplicit but available information about the knowledge model of a correlation engine thus provides important advantages.

The implicit knowledge of the eAutomation model is explicitly modeled through rules. In the previous example for the collocation relationship, the symmetry axiom may be exploited when searching for information. Without the definition of this axiom, searching for collocated resources might depend on the way meta-data was provided for the resources. If one defines that some resource X is collocated with some other resource Y, there is no possibility (without programming or explicit specification) to find out that the resource Y is collocated with resource X. Further, it is impossible to conclude that the resources X and Y cannot be anticollocated with respect to each other.

A transitive axiom is a part of the standard set of axioms. For example, if resource A should start after resource B, and resource B should start after resource C, then resource A should start after resource C as well. Consequently, the property startAfter is defined as a transitive property.

The additional advantage of this approach is that the user can specify arbitrary attributes and relationships and their semantics in a declarative way. For example, the user can extend the model with the new property startBefore and may define that this relationship is transitive. Moreover, he can specify that this new relationship has an inverse relationship to the startAfter property which is already a part of the model.

The set of rules modeling dependencies between the properties of resources is specified in the general rules of an ontology. For example, the fact that the state of a resource group aggregates the states of all its members can be modeled as a rule. General rules are also useful for modeling behavior related to the relationships among resources. For example, the startAfter relationship is used to ensure that a source resource is started only when its target resources are online.

Figure 7 shows the result of inferencing for an ontology containing three resources: R1, R2, and R3. A startAfter relationship can be found between the resources R1 and R2 and between the resources R2 and R3. When the resource R1 has to be started, then because of the startAfter relationship, the resource R3 must be started first. After the current state of the resource R3 has changed to online, the system calculates that the state of the resources R2 and subsequently R1 must be set to online as well. More precisely, the change of the current state triggers the start action, and then the dependency graph is traversed to find R1. This is done because the relationships are defined with appropriate transitive axioms. Thus background knowledge about the domain can be used to find new but implicitly stated facts.

The chain of rules evaluated during an inference process can be represented as a derivation tree.[25] Such a proof structure enables justification of why and how a solution was derived. In the previous example, the derivation tree explains that the desired online state of the resource R2 causes the online status of the resource R1. This change is caused in turn by the online status of the resource R3. Moreover, this proof structure can be used also for calculating the relevance of a particular solution, as we discuss later.

There are several ways to model resource groups. Each model has its own advantages and disadvantages. The choice depends on the type of the application. Figure 8 shows two different ways to model resource groups. The first version, model A, does not contain an explicit concept for resource groups. The information regarding whether some resource is a resource group must be obtained through inferencing.

Because a resource group is an automated resource which contains a collection of resources that are automated as one logical entity, this behavior must be defined using rules. Here the ontology model is more compact and flexible because all rules that are specified for resources can be applied to resource groups as well. The disadvantage is that resource-group-spe-

cific attributes cannot be defined because there is no concept representing groups.

Model B in Figure 8 contains the concept ResourceGroup, which can include either resources or resource groups. (The property includes has two domain concepts: the concept Resource and the concept ResourceGroup.) This solution requires a set of additional rules to model how modification of the state of a group causes a change of the status of the resources within the group. On the other hand, resource groups can be described more precisely. In our approach we selected this latter solution because the model is more understandable for users.

The eAutomation model also allows modeling of the equivalency between resources. In the eAutomation ontology, equivalency is modeled as a separate concept, and associated rules are used to specify consistency constraints, that is, to verify whether an equivalency may be created.

Advantages of ontology-based correlation engines

In this section we discuss the advantages of applying ontologies to correlation engines with our approach. We begin by describing the benefits from a modeling point of view. Then we discuss the benefits from a usage point of view.

Modeling benefits. Inference mechanisms for the deduction of information not explicitly asserted are the most important characteristics of ontology-based systems, as described previously. Here we discuss other advantages of translating the model of the eAutomation correlation engine into the eAutomation ontology. They include the following:

1. *Reusability*—When an automation system has to be applied in a new domain, a model for that domain is needed. It is inefficient and error prone to always build the model from the ground up. Obviously it is better to build these models by reusing smaller well-defined components. An ontology can reuse concept and instance definitions from other ontologies through modularization.[26] This allows the creation of a library of ontologies. The library should contain ontologies that are well separated and coherent with respect to functionality. Users can thus develop their own ontologies by taking advantage of the predefined ontologies from the library, without having to develop the underlying model manually. A library of reusable modules reduces the time and cost of developing a correlation engine. Moreover, it increases the quality of the resulting models as well.

   For example, to model a domain that includes many application servers, one can reuse the Application Server ontology. This ontology includes the Server ontology, which contains concepts, properties, and rules that are characteristic of all servers, independent of the type of service they provide. The eAutomation ontology defined previously could be reused in other ontologies that model autonomic computing systems. The Server ontology includes this ontology directly. However, the eAutomation ontology is indirectly included in the Application Server ontology through the transitivity of the ontology inclusion relationship. The inclusion graph between ontologies is shown in Figure 9.

2. *Extensibility*—The previous example demonstrates the open-closed principle. Each ontology

Figure 8 Different ways to model a resource group

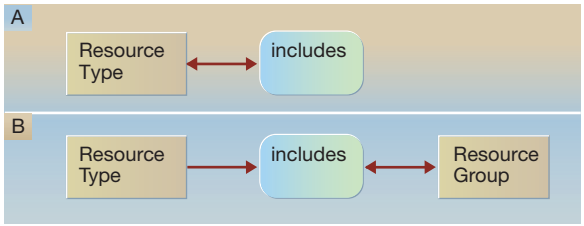
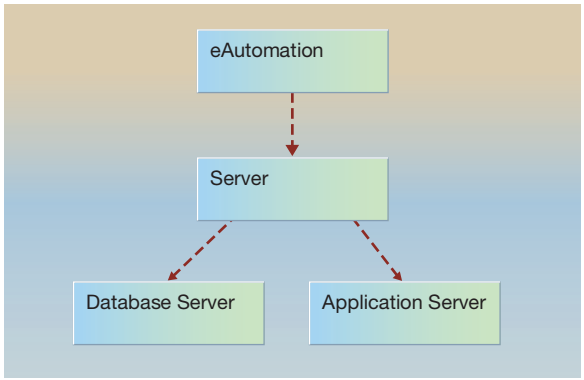


Figure 9 Dependency between ontologies in the service domain. Each node represents one ontology, whereas edges represent inclusion relationships



keeps track of its own information and is a consistent, self-contained, and closed unit. On the other hand, each ontology is open for reuse, in which case any part of its structure can be extended as long as the original model itself is not changed. This means that a new concept which is specific to the current domain can be added very easily as a specialization of an existing concept. For example, the concept WebSphere can be defined as a subconcept of the concept Application Server, which in turn is defined in the previously mentioned Application Server ontology. Moreover, the set of properties can be extended. Because the eAutomation ontology contains only one attribute related to the name of the resource, it is expected that the set of attributes will be extended to cover the features of the concrete domain. For example, the attribute number of threads can be specified for the concept Application Server. The same holds for domain-specific relationships. Thus the attribute load balanced can be a characteristic of the concept WebSphere.

3. *Applicability*—For on demand computing,[2] of which autonomic computing[1,27] is an integral part, it is very important to be able to find a required service. From the viewpoint of the user, there is a need to know the terms or keywords to use when searching for services. Simple keyword queries are valuable when users know what they are searching for and when the information is well-defined. Such queries do not work for on demand computing, where the viewpoints and knowledge levels of the service provider and service consumer may be completely different. Therefore, some mechanism for establishing a shared understanding is needed. Moreover, simple keyword searches cannot deal with synonyms (agent versus actor), abbreviations (World Wide Web versus WWW), different languages (database versus Datenbank) and even morphological variations (point-to-point network versus point to point network), much less context. This problem can be resolved by defining corresponding relations (e.g., synonym, abbreviation) in the domain ontology. Ontology relationships can also be used in the process of navigating through services. For example, it is reasonable to browse from the topic "network" to the topic "protocol."

4. *Verification*—Ontology axioms and rules play an important role in the verification of a model. For example, an axiom that states that two relations are mutually inverse can be used for checking consistency in an instantiated model. Moreover, consider the knowledge that a resource can be a member of either an equivalency or a resource group, but not both of them. This can be modeled explicitly as a rule. Further, the fact that resource groups may not be members of an equivalency can also be represented formally as a rule.

   In general, reasoning tasks can be used to verify a model. Consistency checking, detection of redundancies, and refinement of properties are some of these reasoning activities. Using these concepts it is possible to guide an administrator through the model development process by providing such additional information as why a user's activity did not succeed or what else must be done in order to finish the current activity.[3,4]

5. *Integration*—Heterogeneous information systems can take advantage of the framework of a formal ontology. Applications can become more powerful when they integrate horizontally, but this can require connection of vast amounts of data, legacy systems and custom business applications that may spread across internal operations, partners, suppliers, and customers. There are several reasons for using ontologies for information integration.[9,28] As the most advanced knowledge representation model available today, ontologies can include essentially all currently used data structures. They also can accommodate complexity because the inclusion of deductive logic extends existing mapping and business logic capabilities. Further, ontologies provide shared conceptualization and agreed-upon understanding of a domain, both prerequisites for (semi-) automatic integration.

6. *Evolution*—Since domain models are rarely static, evolution of the model is also important. A model must adapt to changing requirements.[29] Hence, an infrastructure for management of changes that takes into account dependencies between ontologies is necessary.

7. *Visualization*—Visual, hierarchical arrangements of subject categorizations trigger associations and relationships that are not obvious when a model is given only in a textual form. It is well known that visualization is important to the productivity of domain experts.

8. *Open standards*—Any enterprise must connect with other enterprises, any department with other departments, or any one system with other systems. The only realistic way to achieve this requirement is to use open standards. As described previously, ontologies are standard for the Semantic Web. Therefore, the usage of ontologies will enable the compliance of correlation engines with current W3C (World Wide Web Consortium) standards. This will increase the likelihood that models can be exchanged between different systems with minimal loss of knowledge in the translation processes.

Runtime benefits. As noted previously, the resolution of a request in an ontology-based system is realized as an inference process. This process is recorded as a derivation tree, which can be analyzed in order to provide more information about the solutions resulting from an initial request. This tree can be interpreted at the level of the ontology-based correlation engine, where the request corresponds to a problem that has to be resolved and the solution corresponds to an action that must be triggered. In particular, we see three benefits of applying this derivation tree analysis in the context of the correlation engines: (1) justification, (2) ranking, and (3) gap analysis.

1. *Justification* refers to the generation of human-understandable descriptions of the inference process (i.e., how a result was inferred). An explanation of the reasons for suggesting a certain corrective action can be presented to administrators for their information or to a software agent that is responsible for recovering from errors. In this way, the confidence of the administrators in the correlation engine's response can be significantly improved, and the software agent can optimize its actions.

2. *Ranking* involves determination of the relevance of results when many results were inferred. The estimation of the relevance of a result can be obtained by analyzing the complexity of the derivation tree for computing that result. More breadth indicates significant support for that particular result; more depth indicates less confidence in that result.[25]

   This strategy can be applied to assess the importance of proposed actions in autonomic computing systems. An action that is obtained as the result of a larger number of independent rules is more important and therefore should take precedence over actions resulting from fewer independent rules. With respect to tree depth, an action that is recommended directly (i.e. through only one rule) is more important than an action that is suggested indirectly (i.e., through the composition of several rules) because the second case requires that more conditions be fulfilled.

3. *Gap analysis* is related to the discovery of problems or deficiencies in domain knowledge when no result is retrieved. The lack of a result is the most frustrating situation in an information search because users receive no feedback on what was wrong with a request specification. A derivation tree contains information about where a derivation process was broken, what the reasons were, and how it might be possible to continue the process. Therefore, the analysis of a derivation tree can show which parts of the domain knowledge (i.e., rules) are missing, leading to incremental improvement of the knowledge base.

   In order to determine which events to monitor in autonomic computing systems and how to analyze them, administrators must be intimately familiar with the operational parameters of each managed resource and the significance of related events. Certainly, there is no administrator who possesses absolutely all knowledge about a domain. This represents a bottleneck in the development of a model for a concrete domain. The analysis of a broken chain during the inference process can be considered as a methodology for extracting knowledge in a domain in a semiautomatic way. It helps administrators comprehend the effect of an event. If properly used, this process can reduce the number of incorrect activities and can even guide a refinement process. Moreover, this analysis can assist in evaluating whether rules produce the same output as would be generated by a human expert in that domain.

Moreover, information about an inference can be combined with information about the usage of some entities in the domain to discover corrupted resources or problematic situations. This could lead to the development of proactive autonomic computing systems that use predictive methods to eliminate problems or to redirect the system toward better results. In contrast to reactive systems that are able to maintain themselves on demand (i.e., when a problem arises), proactive systems will be able to diag-

nose themselves (i.e., check their states) and plan repairs in advance, eliminating problems before they arise.

Related work

The importance of ontologies for autonomic computing has also been discussed in Reference 30, where the authors applied reasoning mechanisms across not only the entities of the model but also the resource models. Reasoning mechanisms working across resource models could gracefully extend from the autonomic management of a single element, such as an operating system, a database, or a business application, to the autonomic management of a solution, such as a set of components (e.g., operating system, middleware, and application) solving a specific business problem. With respect to modeling, the authors propose the System Management Ontology for representing resources. However, this ontology does not model other aspects of an autonomic computing system, such as events and rules.

In Reference 31 the authors show how current network management methods can be improved by the application of formal ontology techniques. Although they use ontologies for the integration of different models, we have tried to improve the existing models with ontologies. This results in easier integration because all implicit knowledge is represented explicitly.

In References 3 and 4, formal verification methods were applied to the rules of the eAutomation correlation engine to reveal residual bugs resulting in non-termination situations.

Summary

Within the scope of IBM's research and development activities, several correlation engines have been developed and successfully applied in various domains. However, the increasing importance of such self-managed systems in an intra-enterprise environment requires their deep semantic interoperability. Moreover, such integration is of primary importance for on demand computing, one of IBM's strategic goals.

In order to achieve this semantic integration, we have proposed in this paper the usage of ontologies as the conceptual backbones of correlation engines. We see several benefits of using such an approach: First, in the area of domain modeling ontologies facilitate interoperability between correlation engines by providing shared understanding of a problem domain. Second, ontologies provide the formalization of shared understanding necessary to make such understanding machine-processable. Such machine processability is the basis for the next generation of the World Wide Web, the so-called Semantic Web, allowing us to achieve compliance with existing Web standards. Finally, the explicit representation of the semantics of data, in combination with ontologies, enables correlation engines to provide a qualitatively new level of services in autonomic computing systems.

In order to illustrate these benefits, we have presented a case study for the eAutomation correlation engine. We showed how an ontology-based correlation engine can extend the capabilities of the original engine by exploiting the properties of an ontology, namely better modeling of the underlying domain, especially with respect to the expression of axiomatic knowledge, higher reusability, easier extensibility of the system, and the possibility of formal verification of the system.

Finally, we showed how the use of ontologies can help to improve the effectiveness of autonomic systems. In particular, ontologies enable additional levels of services such as justification, ranking of proposed solutions to a failure, and gap analysis.

Acknowledgments

The research presented in this paper would not have been possible without our colleagues and students at the FZI Research Center for Information Technologies and within the IBM Systems Group (ISG) in Boeblingen. It was partially financed by the European Union in the IST-2000-28293 project Ontologging.

Cited references

1. J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer* **36,** No. 1, 41–50 (2003).
2. S. H. Haeckel, "Leading On Demand Businesses – Executives as Architects," *IBM Systems Journal* **42,** No. 3, 405–413 (2003).
3. C. Sinz, T. Lumpp, and W. Küchlin, "Towards a Verification of the Rule-Based Expert System of the IBM SA for OS/390 Automation Manager," *Proceedings of the 2nd Asia-Pacific Conference on Quality Software (APAQS 2001)*, Hong Kong, December 2001, IEEE Computer Society, New York (2001), pp. 367–374.

4. C. Sinz, T. Lumpp, J. Schneider, and W. Küchlin, "Detection of Dynamic Execution Errors in IBM System Automation's Rule-Based Expert System," *Information and Software Technology* **44,** No. 14, 857–873 (2002).
5. eAutomation correlation engine in System Automation for OS/390, http://www.ibm.com/software/tivoli/products/system-automation-390/.
6. System Automation for Multiplatforms, http://www.ibm.com/software/tivoli/products/sys-auto-linux/.
7. T. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition* **5,** No. 2, 199–220 (1993).
8. *Handbook on Ontologies, International Handbooks on Information Systems*, S. Staab and R. Studer, Editors, Springer-Verlag, New York (2004).
9. H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner, "Ontology-Based Integration of Information—A Survey of Existing Approaches," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01) Workshop: Ontologies and Information Sharing*, Seattle, WA, August 4–10, 2001, Morgan Kaufmann Publishers, San Francisco, CA (2001), pp. 108–119.
10. T. Berners-Lee, *What the Semantic Web Can Represent,* http://www.w3.org/DesignIssues/RDFnot.html, 2000.
11. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, D. Fensel, J. A. Hendler, H. Lieberman, W. Wahlster, Editors, MIT Press, Cambridge, MA (2003).
12. A. Maedche and L. Stojanovic, *IBM Correlation Engines,* Technical Report 2-8-04/03, FZI Research Center for Information Technologies, University of Karlsruhe, Karlsruhe, Germany (2003), http://www.fzi.de/wim/publikationen.php?id=1164.
13. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA (1995).
14. eXtensible Markup Language (XML), http://www.w3.org/XML.
15. Resource Description Framework (RDF), http://www.w3.org/RDF/.
16. M. Erdmann and R. Studer, "How to Structure and Access XML Documents with Ontologies," *Data and Knowledge Engineering* **36,** No. 3, 317–335 (2000).
17. XML Schema, http://www.w3.org/XML/Schema.
18. S. Handschuh, S. Staab, and A. Maedche, "CREAM—Creating Relational Metadata with a Component-Based, Ontology-Driven Annotation Framework," *Proceedings of the First International Conference on Knowledge Capture (K-CAP'01)*, Victoria, BC, Canada, October 21–23, 2001, ACM, New York (2001), pp. 76–83.
19. RDF Schema, http://www.w3.org/TR/PR-rdf-schema.
20. Ontology Interchange Language (OIL), http://www.ontoknowledge.org/oil/.
21. DARPA Agent Markup Language (DAML), http://www.daml.org/2001/03/reference.html.
22. Ontology Web Language (OWL), http://www.w3.org/TR/owl-ref/.
23. B. Motik, A. Maedche, and R. Volz, "A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications," *Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002)*, Irvine, CA, October 30–November 1, 2002, *Lecture Notes on Computer Science,* Vol. 2519, Springer-Verlag, New York (2002), pp. 1082–1099.
24. *IBM Tivoli System Automation for Linux on xSeries and zSeries (2nd edition)*, Technical Report SC33-8210-01, IBM Deutschland Entwicklung, Boeblingen, Germany, 2003.
25. N. Stojanovic, R. Studer, and L. Stojanovic, "An Approach for the Ranking of Query Results in the Semantic Web," *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, FL, October 20–23, 2003, *Lecture Notes on Computer Science,* Vol. 2870, Springer-Verlag, New York (2003), pp. 500–516.
26. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, "An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies," *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 20–24, 2003, ACM, New York (2003), pp. 439–448.
27. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era," *IBM Systems Journal* **42,** No. 1, 5–18 (2003).
28. A Maier, H. P. Schnurr, and Y. Sure, "Ontology-Based Information Integration in the Automotive Industry," *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, FL, October 20–23, 2003, *Lecture Notes on Computer Science,* Vol. 2870, Springer-Verlag, New York (2003), pp. 897–912.
29. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, "Ontologies for Enterprise Knowledge Management," *IEEE Intelligent Systems* **18,** No. 2, 26–33 (2003).
30. G. Lanfranchi, P. Della Peruta, A. Perrone, and D. Calvanese, "Toward a New Landscape of Systems Management in an Autonomic Computing Environment," *IBM Systems Journal* **42,** No. 1, 119–129 (2003).
31. J. E. López de Vergara, V. Villagrá, and J. Berrocal, "Semantic Management: Advantages of Using an Ontology-Based Management Information Meta-Model," *Proceedings of the 9th Workshop of the HP OpenView University Association (HP-OVUA),* June 11-13, 2002, http://www.hpovua.org/PUBLICATIONS/PROCEEDINGS/9_HPOVUAWS/Paper_2_1.pdf.

Accepted for publication February 23, 2004.

Ljiljana Stojanovic *FZI-Research Center for Information Technologies, University of Karlsruhe, Haid-und-Neu-Strasse 10-14, 76131 Karlsruhe, Germany (ljiljana.stojanovic@fzi.de).* Ljiljana Stojanovic is a research assistant in the Knowledge Management (WIM) research department at the FZI Research Center for Information Technologies at the University of Karlsruhe. Her research interests include ontology development, evolution, and evaluation, and the use of ontologies for the continuous improvement of knowledge management systems. She received a B.S. degree in electrical engineering in 1994 and an M.S. degree in computer science in 1998, both from the Faculty of Electrical Engineering, University of Nis, Yugoslavia.

Juergen Schneider *IBM Systems and Technology Group, IBM Germany Laboratory, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (jschnei1@de.ibm.com).* Juergen Schneider joined IBM in 1986 and has worked in various development projects as technical leader for IBM zSeries® software. He has over 10 years experience in zSeries systems management, including performance management, operation, and automation. He is currently the lead architect for IBM Tivoli System Automation products, which involve correlation and inference technologies and which provide automated operations and high availability services for all IBM eServer® products. He is a member of several IBM design councils.

Alexander Maedche *FZI-Research Center for Information Technologies, University of Karlsruhe, Haid-und-Neu-Strasse 10-14, 76131 Karlsruhe, Germany (maedche@fzi.de).* Dr. Maedche was department manager of the Knowledge Management (WIM) research department at the FZI Research Center for Information Technologies at the University of Karlsruhe until 2003. He received a diploma in industrial engineering in 1999 from the University of Karlsruhe, majoring in computer science and operations research, and a Ph.D. degree in 2001 in applied informatics, also from the University of Karlsruhe. His research interests include knowledge discovery in data and text, ontology management and learning, and using ontologies in enterprise applications and the Semantic Web. He is a member of IEEE and GI. His homepage at http://www.fzi.de/wim/ contains more information on recent publications, presentations, and events organized.

Susanne Libischer *IBM Systems and Technology Group, IBM Germany Laboratory, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (libische@de.ibm.com).* Mrs. Libischer is manager of eServer Technical Operations and Development at the IBM Germany Laboratory in Boeblingen. In this role she leads several technical support functions, including a data center with several hundred heterogeneous test and development systems. In addition, she is responsible for development projects aiming at improving the manageability of Linux®-based systems. Prior to taking over her current responsibility in 2001, Mrs. Libischer held various technical and management positions in IBM zSeries software development and marketing. These included software test, development, maintenance, and project management functions. Recently, her main focus has been on improving the functionality of IBM's systems management products for zSeries products in the areas of configuration, performance, and workload management, as well as automation. Mrs. Libischer graduated with a bachelor's degree in business administration from the University of Cooperative Education in Stuttgart and joined IBM in 1982.

Rudi Studer *Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB, Universität Karlsruhe, D-76128 Karlsruhe, Germany (www.aifb.uni-karlsruhe.de/WBS).* Dr. Studer is a full professor in Applied Informatics at the University of Karlsruhe, Institute AIFB. His research interests include knowledge management, Semantic Web technologies and applications, Web services, ontology engineering, knowledge discovery and eLearning. He obtained a diploma degree in computer science in 1975, and a Ph.D. degree in mathematics and computer science in 1982, both from the University of Stuttgart. In 1985, he completed his Habilitation in computer science at the University of Stuttgart. From 1977 to 1985, he worked as a research scientist at the University of Stuttgart. From 1985 to 1989, he was a project leader and manager at the IBM Germany Scientific Center. He is director of the Knowledge Management Group at the FZI Research Center for Information Technologies at the University of Karlsruhe (www.fzi.de/wim), a member of the L3S Learning Lab in Hanover (www.learninglab.de), and a co-founder of the company ontoprise GmbH (www.ontoprise.de), which develops semantic technologies. He is president of the Semantic Web Science Association and Editor-in-chief of the journal *Web Semantics: Science, Services, and Agents* on the World Wide Web.

Thomas Lumpp *IBM Germany Laboratory, Schoenaicher Strasse 220, D-71032 Boeblingen, Germany (Thomas.Lumpp@de.ibm.com).* Dr. Lumpp joined IBM in 1999 and has held several different positions in development projects for the System Automation product family. Currently his focus is on the strategic direction of System Automation for Multiplatforms, including an on demand integration. From 1995 to 1999, he worked in research projects on distributed control systems for industrial robots and fieldbus systems. He has received diploma degrees in automation engineering and computer science and a Ph.D. degree in computer science.

Andreas Abecker *FZI-Research Center for Information Technologies, University of Karlsruhe, Haid-und-Neu-Strasse 10-14, 76131 Karlsruhe, Germany (Andreas.Abecker@fzi.de).* Andreas Abecker received a diploma degree in computer science and economics in 1994 from Kaiserslautern University in Germany. He then worked as a researcher, senior consultant, and project manager at DFKI (the German Research Center for Artificial Intelligence) on topics such as knowledge representation, data mining, organizational memory, and business-process-oriented knowledge management. Since 2003 he has managed the Knowledge Management (WIM) department at the Research Center for Information Technology (FZI) at the University of Karlsruhe. His current research interests include the Semantic Web and Semantic Web services, as well as knowledge management and business intelligence applications.

Gerd Breiter *IBM Germany Laboratory, Schoenaicher Strasse 220, D-71032 Boeblingen, Germany (gbreiter@de.ibm.com).* Gerd Breiter is a Senior Technical Staff Member working on the architecture of the IBM on demand infrastructure. Prior to that he worked on the architecture and design of rich media integration for WebSphere®/J2EE® Business Applications and on the architecture and design of transaction and recognition systems in the banking industry. His technical interests involve advanced Web technologies, especially in the areas of Web services and grid computing environments.

John Dinger *IBM Software Group, 3039 Cornwallis Road, Research Triangle Park, North Carolina 27709 (dingerj@us.ibm.com).* Mr. Dinger is a Senior Technical Staff Member in Availability and Business Services Management in the Tivoli Division of the IBM Corporation. Since 2001 he has led architectural development of event correlation, business services, and network management products. Prior to this he worked in the field of systems and network management for telecommunications service providers and traditional enterprise information technology organizations. This included networking communications protocol implementation in both hardware and software. He received a B.S. degree in electrical engineering in 1981 from North Carolina State University.