Model- and Goal-Based Architecture for Situation-Aware Systems

Grit Denker and Carolyn Talcott
SRI International, Menlo Park, California, USA
{grit.denker,carolyn.talcott}@sri.com

January 24, 2004

1 Model and Goal-Based System Design

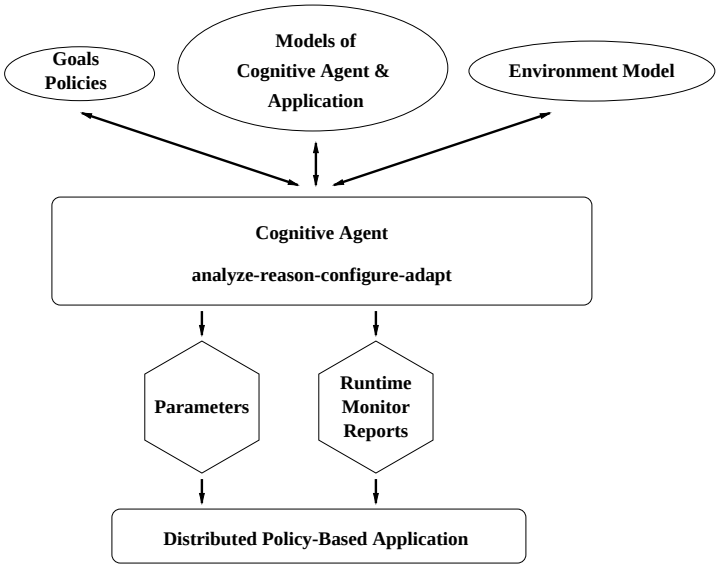

Figure 1: Adaptive System Design

We propose *model- and goal-based system design* as a general engineering principle for distributed, situation-aware systems that can adapt to dynamically changing requirements and environments. A key element of our approach is the use of formal models and their relations with each other and with system parameters and observables. Models used in system design in our framework fall into three main areas:

1. Formal models of the underlying system infrastructure, including models of relevant aspects of devices and of software components of the envisioned system.

2. High-level goals and policies that express various requirements of the envisioned systems, including end-to-end functionality, as well as administrative, computational, or physical distribution requirements. Goals and policies can address performance, security, service classes, quality of service, or other domains of interest.

3. Environment models describing threat and failure models, expected usage patterns, traffic load, or physical environment constraints among others.

This modelling architecture is the basis for our adaptive system design framework. Additional reasoning engines and monitoring services achieve the overall goal of situation-aware and adaptive systems. A reasoning engine analyzes the current situation using its models of system and environment state, compares this with the stated goals and policies, and assigns appropriate initial values to or restrictions on system parameters. Special-purpose, policy-based application software uses these parameter restrictions to compute actual system configurations. Runtime monitors observe the actual execution and inform the reasoning/analyzing engine about ongoing system behavior. If the observed behavior does not fit expectations according to the current system and environment models, adapation of the models is attempted, using justifications for current expectations produced by the reasoning engine. Then reasoning engine recomputes system parameters on the basis of changed models and goals.

2 Framework Applications

We are applying, or proposing to apply, this framework to various application domains, including goal-oriented specification of deep space missions, cognitive networking, and realt-time, embedded systems.

2.1 Goal-oriented Deep Space Mission Design

We came across the idea of a goal-based architecture for distributed system design first in the context of the NASA Mission Data System (MDS). MDS [1] has identified two key ideas for developing a remote agent system that will simplify and reduce the cost of design, test, and operation:

1. A *state-based* approach to system design

2. A *goal-oriented* approach to operation

Every state variable is made explicit along with mathematical models relating state variable values and predicting effects of actions. Sensors monitor observable state and estimators use sensor data and models to determine if goals are being met (part of the reasoning engine).

Mission goals describe the desired outcome of a mission. Goal elaboration rules (part of the reasoning engine) turn high-level goals into temporally constrained executable primitive goals constituting a plan to successfully complete the overall mission. This plan is carried out by scheduling controllers associated with the primitive goals (the policy based application).

We are developing a formal executable model of the MDS framework and goal elaboration mechanisms along with a suite of formal checklists and supporting tools that can be used to achieve greater dependability of goal nets and goal-based operation.

3 Application to Cognitive Networking

In collaboration with J.J. Garcia-Lunes and Brad Smith (University of California, Santa Cruz) we are building on ideas for policy based routing [2] and formal models of networks to address a need for robust, secure and survivable networks, that can be quickly and reliably deployed for applications such as disaster management or network centrict warfare. Challenges include accounting for various quality-of-service (QoS), reliability, security, and other application-level characteristics for the information flow.

By combining formal specification and reflective reasoning with efficient, policy-based routing algorithms, our research will produce a powerful environment for the design and dynamic deployment of self-adaptive, policy-governed network systems that will provide critical functionality under attack.

In particular we propose

- A new approach to network management and control, based on formally specified system models that provide definitions of link characteristics (including performance and constraints on traffic to be carried on a link), expected loads, threat and failure models, and policies addressing security, performance, and resource utilization.

- A self-adaptive *cognitive-layer manager* that maintains these models. Using this knowledge, the manager determines a family of traffic classes, derives possible configurations that will satisfy the policies, and generates link predicates specifying constraints on the traffic allowed in the internet that enforce the desired policies. It provides dynamic adaptation and automatic reconfiguration by accepting policy updates, monitoring network performance and environment states, updating its models, and regenerating link predicates. Using reflection, the cognitive manager has a model of its own behavior. This enables performance evaluation and self-improvement.

- A *policy-based routing protocol* that implements fully distributed routing computations in the context of the constraints specified by the cognitive manager. The routing protocol efficiently computes routes for each traffic class subject to combinations of performance and QoS constraints. Each router generates a forwarding table that implements the computed routes through simple table lookups and independently of the complexity of the traffic classification and routing.

4 Application to Real-time and Embedded Systems

Network-centric, distributed real-time and embedded (DRE) systems are increasingly common and complex, and play critical roles in both civilian and military applications. In collaboration with J. Meseguer (University of Illinois at Urbana-Champaign) and N. Venkatasubramanian (University of California, Irvine), we are planning to apply our general engineering framework to develop a resource aware, adaptive middleware framework to support to DRE systems.

Our objectives include the development of well-defined semantics for the run-time adaptation of middleware services that:

- Ensures safe and correct adaptation such that the QoS requirements of ongoing activities and applications in the system are preserved and problems such as deadlocks, livelocks, and incorrect execution semantics are avoided.
- Provides an understanding of which QoS dimensions (if any) are being sacrificed in the adaptation and to what extent.
- Provides a mechanism for users to negotiate the QoS adaptations with a new set of goals acceptable to the user.

We build on previous experience with formally based middleware for resource provisioning [3]. We propose a Pattern-based Adaptive Resource Management (PARMa) service to support the efficient deployment of DRE applications with guaranteed QoS. PARMa will enable users to express functional and QoS requirements in a goal-oriented fashion, and will configure middleware components, assemble them, and allocate resources to meet QoS goals. In addition, PARMa will monitor system execution and trigger middleware reconfiguration and reallocation of resources as required—for example, if hardware fails. PARMa will be based on:

- formally specified *QoS Patterns* that provide precise documentation of designs,
- a semantic model of the QoS space, and
- models of component QoS properties and resource requirements.

A *goal interpreter* will match user goals against abstract QoS patterns. A *QoS broker* will partially instantiate and configure these patterns, and preselect implementation classes for the components. The resulting annotated configuration will be given to a lower-level *policy-based adaptive resource scheduler* that will complete component customization and allocate system resources. Dynamic adaptation will be performed by first attempting resource reallocation at the lower level, and if this fails, by reconfiguration and redeployment of patterns at the higher levels.

References

[1] D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks. Software Architecture Themes In JPL's Mission Data System. In *IEEE Aerospace Conference, USA*, 2000. `http://techreports.jpl.nasa.gov/1999/99-1886.pdf`.

[2] Bradley R. Smith. *Efficient Policy-Based Routing in the Internet*. PhD thesis, University of California, Santa Cruz, September 2003.

[3] N. Venkatasubramanian, G. Agha, and C. L. Talcott. A formal model for reasoning about adaptive QoS-enabled middleware. In *Formal Methods for Increasing Software Productivity (FME2001)*, 2001. Full version to appear in ACM Transactions on Software Engineering and Methodology, 2004.