Goal-Oriented Approach to Self-Managing Systems Design

Steven J. Bleistein and Pradeep Ray

School of Information Systems, Technology, and Management,
University of New South Wales,
Kensington 2052,
NSW, Australia
sjbl885@cse.unsw.edu.au p.ray@unsw.edu.au

Abstract. This paper is intended to identify issues in design methodology of sophisticated self-managing systems. Whereas self-managing systems are intended to achieve increasingly sophisticated business-level objectives, current agent-oriented design methodologies begin requirements gathering and analysis at much lower levels. This leaves a hazardous gap in the system analysis phase of self-managing system design. This paper proposes *desiderata* for design methodology that include requirements gathering from high-level abstract business goals. It proposes structured and systematic means of decomposing and mapping high-level business objectives down to the finer requirements necessary to agent-oriented methodologies. It explores the possibility of applying experience gained from goal-oriented software engineering techniques as well as methodologies in integrated management of networks and services. Finally, this paper proposes an evolving high-level notation of design patterns to model self-managing systems in an integrated way.

1 Introduction

Self-managing systems have been proposed as the next level of advancement in computer technology. The number of networks, systems, and devices as well as advances in hardware have grown at such an exponential rate that it is no longer possible for human managers to keep pace, nor is the human talent pool sufficient to supply the managers. Self-managing systems are an artificial intelligence solution that proposes to fill the need for management of highly complex networks autonomously, relieving the demand for human intervention, maintenance, and management.[5]

While autonomic computing is meant to achieve increasingly higher-level business objectives in a sophisticated manner, design methodologies of self-managing systems have not kept pace. Systems requirement gathering in current methodologies presumes requirements at lower-levels of abstraction to be given, and do not provide a means for gathering and validating requirements from higher-level business objectives. As the business objectives of self-managing systems become increasingly ambitious and move upward into the strategic-level of organizations, being able to design systems in alignment with high-level abstract business goals becomes critical for success.

The paper is organized as follows: it starts with the definition of self-managing systems in section 2. It then moves on to a discussion of agent-oriented methodologies and their shortcomings in section 3. Section 4 discusses a goal-oriented approach to agent-oriented design to "fill the gap," and defines a set of *desiderata* for goal-oriented design methodology. Section 5 discusses a framework for goal-oriented methodology proposing a high-level, layered design pattern and some techniques borrowed from goal-oriented methodologies in software requirements engineering. Sections 6 illustrates an example of a goal-oriented approach to design of self-managing systems.

2 Self-Managing Systems Defined

Self-managing systems are computing systems that can manage themselves according to high-level objectives that system administrators define for them.[8] IBM refers to these kinds of systems as "autonomous computing" [2] citing four aspects self-management: self-configuration, self-optimization, self-healing, and self-protection. [3] For the present, these aspects are treated as distinct solutions of self-managing systems. In the future, IBM foresees gradual steps of evolutions in self-managing systems in which the distinctions become blurred, ultimately becoming general properties of "self-maintenance" in a general architecture.[8]

IBM envisions the application of this technology in highly strategic ways, beyond the vision of network maintenance. The IBM Autonomic Computing Manifesto cites examples in mass distribution retail, healthcare, and e-services.[5]

Self-managing Systems are in fact interactive collections of autonomous "elements"[8], or more precisely, autonomous *agents*. These agents manage their internal behavior and relationships with other autonomous agents in accordance with policies established by human managers or other agents. The behaviors of autonomous elements are programmed in higher-level, goal-oriented terms. This leaves agents with the responsibility to resolve details of behaviors, relationships, and connections on the fly.[13] Therefore, design of self-managing system in part depends on agent-oriented design methodologies.

This paper proposes a paradigm for design methodology of self-managing based on goal-oriented methodologies, as well as evolving, high-level design patterns. This paper lays out desiderata for such a methodology, and explores recent techniques in software requirements engineering, goal modeling, and integrated management of networks and services.

3 Agent-Oriented Design Methodologies for Self-Managing Systems

At the conceptual level, current agent-oriented methodologies tend to start requirements gathering at a lower level with respect to business objectives and goals of the proposed solution. They tend to focus primarily on individual agent modeling, and then move one layer above to model the interaction between agents, in what is called "society" modeling.

Most of the methodologies are in fact extensions of object-oriented software engineering methodologies, and thus tend to resemble those methodologies in various ways.[6] In terms of agent modeling, the focus is on the roles and responsibilities of individual agents. A higher "societal-level" approach treats the interactions of the agents in the system primarily in terms of individual roles and responsibilities like an *organization*.[14]

Some of the methodologies claim to be "top-down" in terms of modeling "societies" of interactive agents within them based on the individual roles and responsibilities of the agents. Essentially societies are decomposed top-down into agents. The roles and responsibilities of the agents are meant to satisfy higher-level engineering requirements that presumably support the business-level goals.

However, "top-down" implies only a direction of analysis. It implies nothing about the *starting point.* While the methodologies may in fact be top-down in terms of the direction of the analysis, they rely on a fairly well defined set of requirements at the outset. They do little to support design when the starting set of requirements is in terms of much higher business-level goals and objectives. Therefore, there is a gap in the current design methodologies for sophisticated self-managing systems in terms of aligning system requirements with high-level business strategies and objectives.

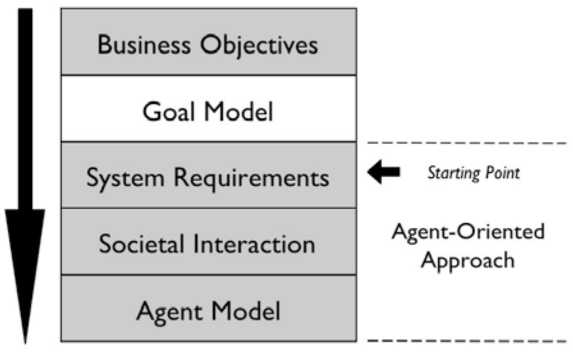

Figure 1 The Gap in Top-down Analysis Levels of Self-Managing Systems

3.1 The Dilemma with Current Methodologies

How should the system designer handle requirements that are given in much higher-level, abstract, business terms? How should the system designer validate requirement alignment with overall business objectives?

Agent-oriented methodologies tend not to provide a structured and systematic means of discovering business-level goals much less decomposing them down to an agent-based "societal" level model. Moreover, these methodologies also tend not to provide a clear means of linking the agent-models, consisting of roles and responsibilities, back up to high-level business goals for validating purposes. The dangers in leaving these aspects out of the design methodology are as follows:

- Lack of top-down design methodology from the highest levels runs the risk of producing an incomplete society-level model.
- Lack of clear traceability of agent models to the higher-level business goals they are intended to support makes it difficult to validate a system design in terms of satisfying business-level objectives.
- Lack of agent-role to business-goal traceability makes it difficult to provide a framework of evaluation of system design alternatives in terms of satisfying overall business-level requirements.

These risks are realized in terms of failed projects or results misaligned with business objectives.

3.2 Pitfalls in Agent Projects

Wooldridge and Jennings cite an number of pitfalls in agent-oriented development through their accumulated experience with agent projects.[15] Among these pitfalls are a number that are arguably the consequence of a lack of business-goal orientation during the analysis and design phases, two of which are presented below.

3.2.1 Not Understanding the Objectives of the Project

Here the authors cite "You don't understanding why you want agents." They observe managers "initiating projects without clear goals." They refer to managers' being seduced by "big promises" in IT literature, as well as their lacking "clear visions of how agents will enhance existing product lines or generate new ones." To describe the author's experience in more general terms, these observations are attributable to lack of formal framework for evaluation in terms of business objectives, as well as a lack of framework to support project requirement alignment with business goals.

3.2.2 Not Understanding How to Apply Agents

Here the authors cite "You don't know what agents are good for." The authors claim that this pitfall is related to the one above in that it "concerns a general lack of clarity of the purpose of the technology and the lack of understanding about the degree of its applicability." They cite observation of IT departments, energized from having developed one or two agents successfully, to begin looking for other applications, in a kind of *ad-hoc* solution-looking-for-a-problem type of strategy. They argue that this invariably "…leads to mismatches and dissatisfaction, either because the full potential of what an agent could add to the application is not achieved (because agents have the wrong functionality or emphasis), or else because only a subset of the agent's capabilities get exploited…"

This kind of pitfall is clearly a case of bottom-up analysis of requirements, with neither a complete picture of business goals nor a clear understanding of the links to higher-level business objectives. There is a lack of linkage analysis between the agent model and higher-level objectives for the purpose of validation.

The authors vigorously discourage managers from this kind of arbitrary approach. They encourage understanding of "…how and where your new technology can most usefully be applied." The authors' advice goes beyond simply understanding the technology on a technical level. It is in essence advice to understand the requirements in terms of the high-level business needs of a complete system, and how agents are applied to those needs.

4 Goal Oriented Design of Self-Managing Systems

Self-managing, autonomic computing systems are in essence goal-oriented. They maintain themselves according to high-level, overall business goals. The intent behind autonomic computing is to free systems administrators from the details of systems operations and maintenance, and to provide users with machines that run at peak performance continuously.[8]

This intent, expressed in terms of autonomous integrated management of networks and services, is in fact nothing less than a high-level, abstract business goal. Realizing this business goal is a matter of decomposing it into smaller subgoals. At the most granular level, the subgoals are allocated to autonomous agents. In this respect, the complete discovery of business-level goals, their correct decomposition, and proper allocation to autonomous agents is critical to the design of well-functioning self-managing systems that support the higher-level business objectives.

Therefore, this paper proposes that a goal-oriented approach to design of self-managing systems as means to bridge the gap between high-level business objectives and agent-oriented design methodologies.

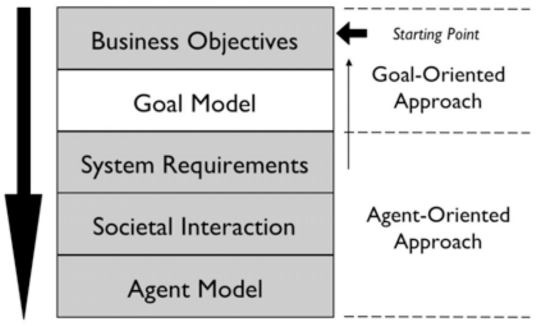

Figure 2 Goal-Oriented Approach to Bridge the Gap Downward

4.1 Goal-Oriented Methodologies

In recent years, a number of advances have been made in the field of goal modeling in terms of paradigms and methodologies, primarily as applied to the field of software requirements engineering. This paper argues that parts of these methodologies have application in the design of self-managing systems. This is an active area of research, and this assertion is still subject to validation.

4.2 Concepts and Terminology

Before suggesting a possible set *desiderata* for such a methodology, let us examine some terminology in goal modeling. Goal-oriented modeling and software requirements engineering have some specific terminology defined here that will be used in the rest of this paper.

4.2.1 Functional vs. Non-Functional Requirements

Most software engineers are familiar with the concept of functional requirements (FR). In modeling system requirements, non-functional requirements (NFR) are often just as important, and can make or break a project depending on how well they are satisfied. NFRs are defined as system requirements that are not tied to any particular functionality.[1] Examples of non-functional requirements include system-wide requirements such as performance, accessibility, and security. In this respect, modeling NFRs is relevant to modeling requirements of self-managing systems in that the four aspects cited by IBM of self-configuring, self-optimizing, self-healing, and self-protecting are all in fact non-functional requirements by this definition.

4.2.2 Hardgoals vs. Softgoals

Goals are distinguished between hard and soft. The distinction between the two is that of measurability. Hardgoals are satisfied when measurable criteria have been achieved. Softgoals however, have no concrete measurable criteria. Their status is thus "fuzzy." Instead of discussing softgoals in terms of being satisfied, which implies an absolute completeness regarding goal achievement, softgoals are "satisficed." [1] Similar to the use of the term in domain of decision support systems, the nuance is that a goal has been satisfied to a degree that is "good enough" to accept.

The importance in the distinction between hard and soft goals is that NFRs are often represented in terms of softgoals.[1] Moreover, high-level business goals are very "soft" in nature. Therefore, representation and treatment of softgoals is important in design of self-managing systems.

4.2.3 Tasks vs. Operationalizations

Hardgoals at the lowest level are satisfied via tasks. Processes are a representation of knowledge organized into the design of an action whose results are intended to satisfy or contribute to a goal.[12] Processes define tasks. The end result of a process is achievement of or toward a goal. Softgoals at the lowest level are satisfied by "operationalizations." The distinction here is that agents do not necessarily achieve goals by a fixed set of tasks. Agents modify their behavior according to the environment, motivated by achievement of goals assigned to them.[10] The specific behavior required to achieve such goals is not known until run-time.[16]

4.3 *Desiderata* in Goal-Oriented Approach to Agent-Oriented Design Methodology

The following proposed *desiderata* are based on commonality of practices in goal-oriented software requirements engineering methodologies[1] [17] [18] as well as specific practices that have applicability to resolving design issues mentioned above.

- Systematic, structured goal and requirement discovery methods from the highest level to ensure Completeness of goal model
- Framework for modeling non-functional requirements as well as "soft" objectives
- Support of goal decomposition from high-level business goals down to the level of requirements necessary to agent-oriented design methodologies
- Traceability from agent-models back up to high-level business goals to validate completeness and goal alignment of the self-managing system requirements

5 Proposed Framework for Goal-Oriented, Self-Managing System Design

In recent years, a number of sophisticated tools in the field of goal modeling of business requirements applied to software requirements engineering have been developed. These include *L'Ecritoire* as part of the ESPRIT CREWS Project[17], non-functional requirements engineering using "softgoals,"[1] and Goal-Oriented Requirements Language (GRL). These are elaborate and sophisticated tools for eliciting and modeling of software engineering requirements of complex systems from a high level, but are not completely appropriate to *desiderata* cited above. Experience from each of these approaches, however, yields insight into good and novel practices that can serve as part of the framework for goal-oriented design of self-managing systems.

5.1 Modeling Solution Patterns from Top-Down

Many software requirements engineering methodologies discuss the need to model requirements from business goals. Business goals tend to be more abstract in nature than functional and non-functional software requirements when considered in the context of an information system. It is the job of the systems designer to translate the business-level abstraction to concrete logical architectures. This can be a cumbersome and confusing process because it requires bridging two complex and disparate domains: information systems and business management. Often, this results in communication breakdowns and fluctuating requirements.[11] This leads ultimately to misaligned results, failed projects, and frustration among business managers and engineers alike.

To address problems such as these in the field of electronic commerce, researchers at IBM have proposed e-business design patterns for reliable, robust, reusable component solutions for common issues in e-business just as in software engineering.[4] IBM proposes patterns that address the solution at different layers from (highest to lowest) "business integration and composite patterns" to "application patterns" and then to "runtime patterns."

Ray has generalized this concept to use layered design patterns as a means of developing integrated management solutions for e-business networks and services, mapping solutions from the high-level *business management pattern* layer, down through the *logical pattern* layer describing e-business management applications, and ultimately down to the platform or *physical layer*.[13] The mapping of the IBM layers to Ray's generalized framework is illustrated in Figure 3 below.

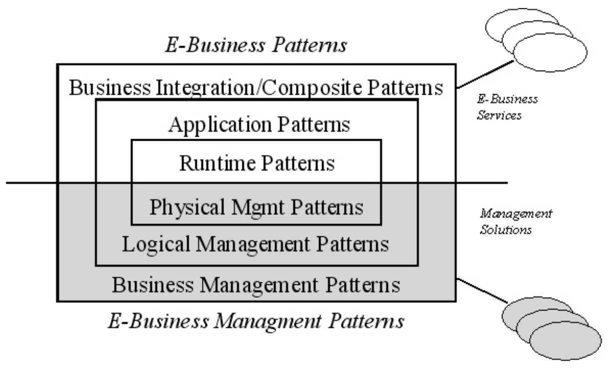
E-Business Patterns
Business Integration/Composite Patterns
Application Patterns
Runtime Patterns
Physical Mgmt Patterns
Logical Management Patterns
Business Management Patterns
E-Business Managment Patterns
E-Business Services
Management Solutions

Figure 3 E-Business Management Patterns[13]

5.2 Developing a Complete Top-Down Goal Model

The e-business pattern framework proposes a solution integrating business-level, logical-level, and physical-level representation as a means of aligning solutions implementations at all levels. While it is not the intention of this paper to propose specific design patterns for self-managing systems, this paper builds on Ray's design pattern concept in terms integrating design solutions from business-objectives down. Therefore, this paper thus proposes a new paradigm in design patterns for self-managing systems illustrated in Figure 4.

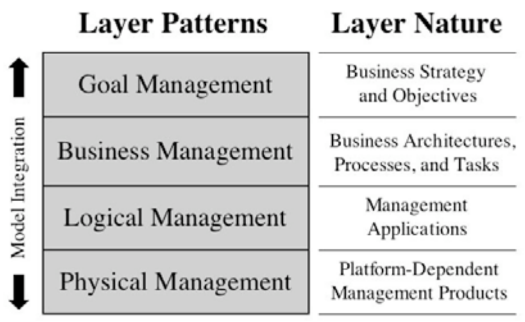

Figure 4 Layers of Top-Down Integrated Design for Self-managing Systems

This paper proposes achieving integration of design pattern models at different layers though the following:

- Discovering and validating business-objective requirements through goal elicitation
- Mapping higher-level goals and requirements to lower-level ones through systematic and structured decomposition
- Realizing a complete goal model integrated at all levels for purposes of requirement validation and alignment with business objectives

5.3 Analysis from the Top

Without identifying goals at the business management layer in a complete way, it is not possible to validate with a high-level of certainty the proposed solutions at the logical and architectural layers. In essence, goal decomposition is a tree-like structure. If a systems analyst identifies requirements at the lower levels and then maps upward toward higher level goals, he may be able to validate that his lower level requirement does indeed support the higher-level objectives; however, without performing a top-down analysis beginning with the higher-level goals, the business analysts risks developing an incomplete model at the lower levels. In the end, this leads to incomplete or misunderstood requirements.

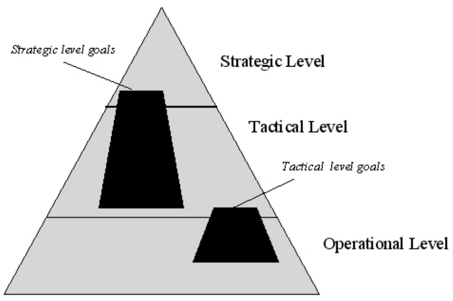

Figure 5 Solution Positioning in Strategic Hierarchy

5.4 Goal Hierarchies in Business Strategy

Goal modeling must begin at the highest business goal level of a project. Most methodologies tend to deal with requirements at a lower level than in the area of business strategy. It is important to know where a proposed solution positions itself in terms of the strategy hierarchy of a business. In this way, it is easier to understand where the highest-level business objectives of the solution scope exists so as to know where to begin the top-down goal analysis and decomposition (see Figure 5).

5.5 Distinguishing Between the Strategic, Non-Strategic, and Tactical

Some simple guiding questions can also help situate the level of the proposed project or solution:

- Is the proposed solution supporting business strategy, or is the proposed solution *the* business strategy?
- Is the proposed solution supporting the core business, or is the solution part of the value chain, processed, and sold as a service to customers?

The above distinctions may seem abstract, so let us take an example:

IBM discusses paradigms of self-managing systems of different business levels. One paradigm is in terms of automatic maintenance and management of networks as discussed above[4]; however, IBM also discusses strategic-level examples in its Autonomic Computing Manifesto.[5]

In this example, IBM illustrates an “e-sourcing provider” using autonomic computing as an integral part of the services the company is selling. Thus, the requirements of the self-managing system originate at the highest level of the company’s strategy, and are in fact intertwined with it. A change to the self-managing system would mean a fundamental change to the core business of the company, and in essence a change to the services it is selling to its customers.

Compare this case to a company that is using a self-managing system for its intranet as a means of more efficient network management. A change to this system might require hiring of more human managers, but it does not change the core business of the company.

In the e-sourcing provider example, understanding the goals of the business strategy is the same as understanding the requirements of the self-managing system. Every lower-level goal down to the logical and architectural requirements ought to be traceable back up to supporting a strategic-level business goal. Thus, understanding the strategic-level business goals of the self-managing system at the beginning of the design process is absolutely critical to the success of the project, and indeed to the success of the business.

5.6 Systematic Goal Discovery

There has been a large amount of literature on business goal discovery through scenario analysis,[7] which this paper will not cover. Scenario analysis originated in methodologies in human-computer-interaction and computer-supported cooperative work. Since that time, use case analysis has become standard practice in software design and development using UML, and more recently, use-case mappings. Use case mappings are a more sophisticated extension of UML use cases[20] that have been proposed in combination with GRL as a standard for User Requirements Notation currently under consideration by the ITU-T.[9] Eliciting goals through scenarios has many advantages that will not be treated here, but the primary disadvantage of this process is that it is in effect bottom-up.

A scenario is an instance of a process. Processes are a representation of knowledge organized into the design of an action whose results are intended to satisfy or contribute to a goal.[12] Scenario analysis is in effect a means of extrapolating goals from instances of processes; however, this bottom-up methodology neither ensures completeness nor validity of a goal model.

The CREWS *L'Ecritoire* project addressed this issue directly with a methodology that couples goals with scenarios. *L'Ecritoire* provides a means to check validity and completeness through a systematic *bidirectional* analysis of goals and scenarios, in this case meaning top-down *and* bottom-up. Essentially, the systems analyst uses scenarios to elicit goals, and then uses goals to validate scenarios. Goal analysis can also be used to elicit further scenarios, and so on.[17] In this way it is possible to build a complete goal model and validate it. Therefore, in terms of satisfying the *desiderata* above, this paper proposes adoption a similar approach: goal-scenario coupling, bidirectional analysis, and discovery of both goals and scenarios from each other.

5.7 Goal Decomposition Patterns

When eliciting and discovering goals, it is important to understand what kind of goals they are and where they are situated in relation to others. Subgoals are called *refinements*. While the discovery supergoals from subgoals is matter of asking questions like "Why?", the discovery of *refinements* from goals is a matter of asking questions of "How?" Among refinements, there exist two types of patterns: *alternative* and *compositional*.

Compositional patterns consist of multiple refinement goals that must be satisfied in order to satisfy a supergoal (Figure 6). Discovering composite goals involves asking questions like "What else is necessary?"

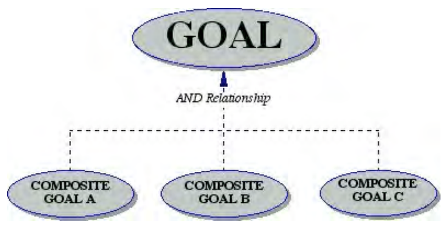

Figure 6 Alternative Goal Pattern

Alternative goal patterns consist of subgoals, any of which may be achieved to satisfy a supergoal (Figure 7). Discovering alternative goal patterns invoices asking questions like "How else can this be achieved?"

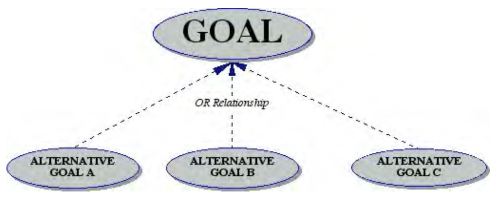

Figure 7 Compositional Goal Pattern

5.8 Handling of Softgoals

L'Ecritoire as a tool is also able to handle the concept of a "softgoal" like GRL; however, its approach to softgoals is different. *L'Ecritoire* aims at continual decomposition of softgoals into increasingly concrete refinements until they are reduce to concrete elements.[17] By contrast, GRL and non-functional requirements engineering seem to accept the some softgoals will always be soft and impossible to reduce to concrete subgoals.

While this is an open area of research, it appears that no matter how "soft" a goal may be, it is always possible to decompose it a bit more as *L'Ecritoire* presumes. For the purposes of the desiderata above, this paper proposes adoption of this concept for handling the more abstract goals.

5.9 Allocation of Goals to Autonomous Agents in a Self-Managing System

The set of modeling tools in non-functional requirements engineering[1] and GRL[9] represents goals in terms of both hardgoals and softgoals as intentional elements. It also includes tasks and resources as elements, and an array of modeling the relationships between the elements. [19]

This is fine for conventional software engineering purposes; however, for the purposes of decomposing goals for self-managing systems, goal decomposition down to the finest level just above task or operationalization ought to be sufficient to lay the groundwork for goal allocation to autonomous agents.

As current methodologies in agent modeling define agents in terms of roles and responsibilities, the responsibilities can be allocated in terms of the refined goals. The collection of responsibilities of an agent defines its role. In this way, the purpose of an agent in a self-managing system can be traced back up through the goal structure all the way to the strategic-level business objectives.

6 Illustrated Example: e-Sourcing Provider

Let us return our attention to the IBM's example of the e-sourcing provider who is acting as a service broker. [5] Below is a goal model of system. This example is far from complete and exhaustive, but it is intended to illustrate some of the goal-oriented concepts of this paper. In this example, a generic notation for goal modeling is being used rather than the more sophisticated GRL for simplicity.

6.1 Goal Discovery and Modeling

Let us take a fairly simple strategic goal of "Double sales volume of services." For simplicity, the goal model has been intentionally limited. We can begin refining this goal by asking "How?" questions and developing scenarios. In interviews with stakeholders, a number of scenarios for increasing customers might arise. It would be up to the systems analyst to recognize the commonality in the scenarios and elicit a goal of *increasing* customer base; however, analyzing the goal coupled with the scenarios again downward, the systems analyst may discover alternative goals for *maintaining* and *acquiring* customers. Further decomposition and scenario analysis yields more refined goals as illustrated. Ultimately, the designer arrives at goals that are so granular that further refinement would mean defining tasks and operationalizations. These goals are filled in white below. This is where the systems analyst stops.

6.2 From Goal Model to Agent Model

The final subgoals are as follows:

- Negotiate offer price (from service providers)
- Find providers
- Validate credit and performance history (of providers)
- Negotiate SLA
- Enforce SLA

Each of these subgoals can be traced upward to supporting higher-level goals of the overall strategic objective of the system.

The end subgoals in white also represent the collection of responsibilities to be allocated to the autonomous elements of the self-managing system. For example, we could allocate roles and responsibilities as follows:

- Role: Provider Negotiator – responsible for (1) Negotiating SLA, (2) Negotiating price
- Role: Provider Sourcer – responsible for (1) Finding providers, (2) Checking credit and performance history
- Role: Contract Enforcer – responsible for SLA Enforcement

Perhaps we would want to split the Sourcer role into a Finder agent to find providers, and a Risk Manager to check the suppliers found, but at this point we are passing into agent-oriented methodology.

Thus, goal modeling does indeed enable decomposition from high-level abstract business objectives to autonomous agent roles and responsibilities at the top of the agent-oriented design methodology.

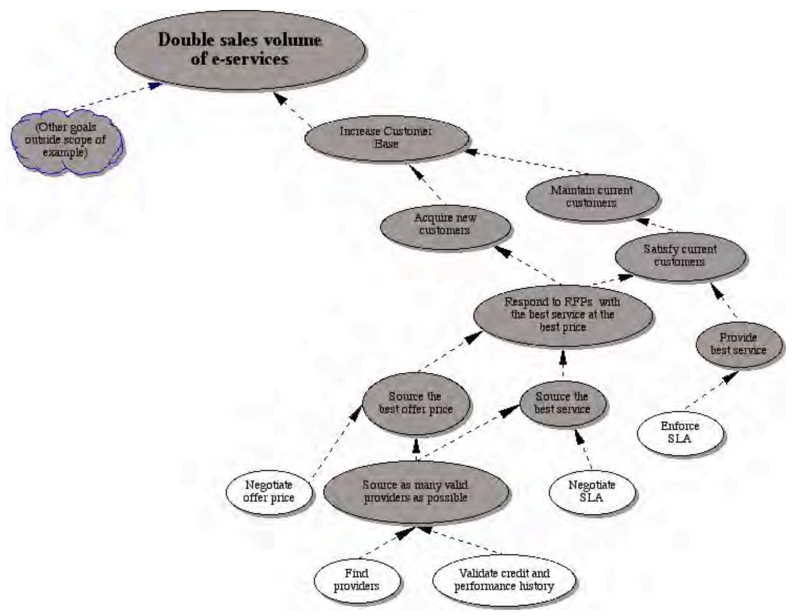
Double sales volume of e-services
(Other goals outside scope of example)
Increase Customer Base
Maintain current customers
Acquire new customers
Satisfy current customers
Respond to RFPs with the best service at the best price
Provide best service
Source the best offer price
Source the best service
Enforce SLA
Negotiate offer price
Source as many valid providers as possible
Negotiate SLA
Find providers
Validate credit and performance history

Figure 8 Partial Goal Model for e-Service Provider Example

7 Conclusion

As self-managing systems become ever more sophisticated, and as their uses increasingly enter the realm of business strategy, there is a need to link design methodologies to higher-level business objectives. This paper proposes that this end is achievable through a goal-oriented approach. Moreover, this paper proposes evolving, high-level design patterns to integrate solutions from the goal, business, logical, down to physical pattern layers. Having defined a set of *desiderata* for a goal-oriented design methodology for self-managing systems, this paper illustrates an approach that leverages experience gained from goal modeling techniques as applied to software requirements engineering. Finally, this paper illustrates how such a goal-oriented method would be applied, linking lower-level systems requirements to high-level business objectives of self-managing systems.

References

1. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional Requirements Software Engineering. Kluwer Academic Publishers, Boston, Dordrecht, London, 2000.

2. IBM Corporation: Autonomic Computing: Creating Self-Managing Systems. URL: http://www-3.ibm.com/autonomic/index.shtml.

3. IBM corporation: About IBM Autonomic Computing: Self-Managing Systems. URL: http://www-3.ibm.com/autonomic/selfmanage.shtml.

4. IBM corporation: IBM Patterns for E-Business.
URL: http://www-106.ibm.com/developerworks/patterns/.

5. IBM corporation: Autonomic Computing: IBM's Perspective on the State of Autonomic Computing. URL:http://www.research.ibm.com/autonomic/manifesto/.

6. Iglesias, C., Garijo, M., Gonzalez, J.: A Survey of Agent-Oriented Methodologies. Intelligent Agents V-Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg. 1999.

7. Jarke, M., Kurki-Suonio, R.: Guest Editorial: Introduction to the Special Issue. IEEE Transactions on Software Engineering, Vol. 24, No. 12, December, 1998.

8. Kephart, J., Chess, D.: The Vision of Autonomic Computing. IEEE Computer, Volume 36, Issue 1, January, 2003.

9. Lin, L, Yu, E.: From Requirements to Architectural Design - Using Goals and Scenarios.
ICSE-2001 Workshop: From Software Requirements to Architectures (STRAW 2001), Toronto, Canada, pp. 22-30, May 2001.

10. Luck, M., d'Inverno, M.: A Formal Framework for Agency and Autonomy. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 1995.

11. Nell, C.: Business and Systems Development: Opportunities for an Integrated Way-of-Working. Perspectives on Business Modeling, pp. 197-212, Springer Verlag Berlin, Heidelberg, New York, 1999.

12. Ray, P.: Integrated Management from E-Business Perspective. International Kluwer Academic/ Plenum Publishers (NY, Boston. Dordecht, London, Moscow), ISBN: 0-306-47485-9, Jan 2003.

13. Ray, P.: Design Patterns for the Management of E-Business Networks and Services. Preprint submitted to Elsevier Science, February 7, 2003.

14. Wooldridge, M.J., Jennings N.R., Kenny, D.: A methodology for agent -oriented analysis and design. Proceedings of the Third International Conference on Autonomous Agents, 1999.

15. Wooldridge, M., Jennings N: Pitfalls of Agent-Oriented Development. Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98), 1998.

16. Yu, E: Agent-Oriented Modelling: Software Versus the World. Agent-Oriented Software Engineering, AOSE-2001 Workshop Proceedings., LNCS 2222 . Springer Verlag. pp. 206-225, 2001.

17. Rolland, C., Souveyet, C., Ben Achour, C.: Guiding Goal Modeling Using Scenarios. IEEE Transactions Software Engineering, Vol. 24, No. 12, December, 1998.

18. Bubenko, J., Rolland, C., Loucopoulos, P., DeAntonellis, V.: Facilitating "Fuzzy to Formal" Requirements Modeling. Requirements Engineering, Proceedings of the First International Conference on Requirements Engineering, 18-22 April, 1994.

19. Goal-Oriented Requirements Language: URL: http://www.cs.toronto.edu/km/GRL/ .

20. Buhr, A.: Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering, Vol. 24, No. 12, December, 1998.