

The AML Approach to Modeling Autonomic Systems

Radovan Cervenka, Dominic Greenwood, and Ivan Trencansky
Whitestein Technologies
Panenska 28, 811 03 Bratislava, Slovakia
Tel +421 (2) 5443-5502, Fax +421 (2) 5443-5512
{rce,dgr,itr}@whitestein.com

Abstract

Autonomic systems are typically distributed, complex and concurrent systems, comprised of multiple interacting autonomic elements that often exhibit emergent behavior. Design and development of such systems is a non-trivial task that by definition requires specific software engineering approaches, including the use of specialized modeling techniques. This paper takes the well-known Prospecting Asteroids Mission from NASA, an autonomous and autonomic system for future exploration of asteroid fields, which can be considered as a axiomatic example of how autonomic principles can be applied in real terms. Introducing the Agent Modeling Language (AML), we demonstrate using a series of didactic examples how AML can be applied to efficiently, accurately and comprehensively model the PAM system. A selection of AML models specifying the PAM domain, goals, architecture, and behaviors are presented which help demonstrate the utility of AML when modeling autonomous and autonomic systems.

1. Introduction

Autonomic Computing is a technological realization of the disposition of natural, biological systems toward maintaining systemic equilibrium conditions through integrated feedback mechanisms. Biologically speaking, this centers on the ability to self-diagnose problems and automatically apply self-healing processes. In technical terms, this core model can be extended to cover other *self*-* mechanisms such as self-assembly, self-configuration, self-optimization, etc. all of which are intended to reduce the need for external intervention by allowing infrastructures to manage themselves.

Accepted for the International Conference on Autonomic and Autonomous Systems (ICAS'06), July 19-21, 2006, Silicon Valley, USA.

However, the very complexity of many systems that lend themselves well to autonomic control can often imply difficulty in designing the autonomic system itself. While biological systems use evolution as a means to derive control mechanisms through emergence over lengthy periods of time, designers of autonomic software-driven systems must focus on designing systems that operate as expected at deployment time, with emergence often used only to tune existing behaviors, rather than create new ones. This implies that useful models need to account for many aspects including distribution, concurrency, autonomous behavior, complex interactions and more, in order to help clarify the design of a system and reduce the chance of inaccuracies, ambiguities and incompleteness occurring at implementation time.

Autonomic systems are intrinsically intended to reduce the complexity of managing systems through automation. It is clear that capturing the logic, behaviors and interactions required to achieve such automation requires quite specialized modeling techniques. Also, while the implementation of autonomic systems can be achieved by many means, the control and management aspects of individual autonomic elements and complete autonomic systems can, and often are, accomplished by means of software agents. The value and relevance of agents in this role has been discussed in several previous papers including [9, 2, 4] and will thus be taken as an assumption for the purposes of this paper.

In an attempt to ameliorate some of the intrinsic design complexity, in this paper we describe an approach to modeling autonomic systems using the AML modeling language, recently created to model software agent systems and applications. AML is a comprehensive visual modeling language that extends UML 2.0 with a variety of important features specific to software agent technology. As we will demonstrate through the course of the paper, we have found that AML lends itself particularly well to visualizing the complex systems designed to employ the principles of Autonomic Computing for the purposes of *self-management*.

Our approach is focused on examining the utility of an agent-oriented modeling language when applied to problems in the autonomic computing domain. Agent-based modeling is a powerful and flexible means of modeling complex, distributed, interconnected and interacting systems and as self-* principles can be enacted through the behaviors of autonomous agents, we expect that a well-specified modeling language should be intrinsically capable of capturing them both visually and logically.

The particular example we employ to illustrate the utility of AML in this domain is the NASA *Prospecting Asteroids Mission (PAM)*. As described in [7], the PAM system is based on a combination of both autonomous and autonomic computation. While these mission types are powerful in concept, their complexity and intrinsic dependence on autonomous and autonomic principles makes them difficult to accurately design and verify. We therefore provide a snapshot of an analytical model for aspects of PAM, focusing on a conceptual architecture and logical model of the system’s primary behaviors.

Throughout the course of this paper we will demonstrate how the core features of AML can be applied in this respect, although due to space limitations a comprehensive description of AML abstract syntax, semantics, and notation is not provided (for details see [1]).

The remainder of this paper is structured as follows: Section 2 provides an introduction to AML. Section 3 describes the PAM system and presents AML models of its domain, top-level system goals, overall operational scenario, and architecture. Section 4 and its subsections demonstrate the usage of AML on modeling selected scenarios used to realize autonomic behavior of the PAM system.

2. Agent Modeling Language

Agent Modeling Language (AML) [1, 6] is a semi-formal visual modeling language for specifying, modeling and documenting systems that incorporate concepts drawn from multi-agent systems theory. AML is a comprehensive and versatile extension to UML 2.0 designed to address the specific qualities offered by multi-agent systems (MAS) that are difficult, or impossible, to model with more traditional modeling languages such as UML. AML can also be applied to other domains such as business systems, social systems, robotics, and of course, autonomic systems. In general, AML can be used whenever it is suitable or useful to build models that (1) consist of a number of autonomous, concurrent and/or asynchronous (possibly proactive) entities, (2) comprise entities that are able to observe and/or interact with their environment, (3) make use of complex interactions and aggregated services, (4) employ social structures, (5) enable goal-based problem decomposition, and (6) capture mental characteristics of entities.

AML incorporates and unifies the most significant concepts from the broadest set of existing multi-agent theories and abstract models, extending them where necessary and assembling the entirety into a consistent framework specified by the AML metamodel. A list of sources is provided in a previous paper [6].

AML supports all of the typical architectural and behavioral concepts associated with multi-agent systems, including entities, communicative interactions, observations and effecting interactions, behavior abstraction and decomposition, social aspects, goal-oriented reasoning, services, ontologies, deployment and mobility. As software agents are capable of assuming the task of operational management and coordination of autonomic elements, it is therefore a logical assumption that AML can be applied to model these and, by extension, the broader characteristics of autonomic systems.

3. Prospecting Asteroid Mission

PAM [7, 8], the Prospecting Asteroid Mission, is an application of the Autonomous NanoTechnology Swarm (ANTS) mission architecture with the aim to explore the asteroid belt for asteroids with certain characteristics. PAM involves the launch of a swarm of pico-class, self-similar, low-power, low weight spacecrafts capable of operating as fully autonomous, yet addressable, adaptable units.

Fig. 1 depicts a class diagram of the physical operational environment of the PAM system. The environment type *Space* comprises several *Objects*. Each *Object* is characterized by *Placement* within the *Space* describing it in terms of position, direction, rotation, speed, etc. Two specialized types of *Objects*, inheriting all its properties, are *Asteroid* and *Spacecraft* (modeled as an abstract AML agent type).

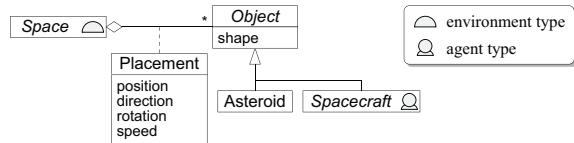


Figure 1. Environment of PAM.

3.1. Overall operational scenario

The overall PAM scenario, modeled as an activity diagram, is depicted in Fig. 2. A transport ship carries a laboratory that assembles the tiny spacecrafts. Once it reaches a certain point in space where gravity forces are balanced, the transport releases the assembled swarm which heads for the asteroid belt. The whole swarm of spacecrafts then travels to the asteroid belt using their solar sails and tiny thrusters.

When the swarm reaches the destination smaller groups of spacecrafts, called sub-swarms, are formed in order to operate autonomously. Each sub-swarm then repeatedly localizes and explores asteroids, independently on other sub-swarms. When the mission is over, the scenario is terminated.

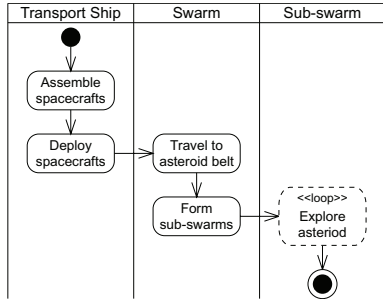


Figure 2. Overall PAM scenario.

3.2. Mission goals

Fig. 3 depicts the decomposition of the overall mission goal, *Prospect asteroids*, into a hierarchy of more fine-grained sub-goals. The goals are connected by means of the contributions representing their logical relationships. For instance: (1) the goals *Detect asteroid*, *Select target*, *Configure teams*, *Move to target*, and *Gather scientific data* are necessary, strongly positive contributions to the goal *Explore asteroid*, or (2) the goals *Protect against solar storm* and *Avoid collision* partially positively (by 90 and 50 percent respectively) and contribute sufficiently to the goal *Secure mission*. Each goal can be characterized in terms of commit-/pre-/cancel-/post-condition and invariant. Also the contribution relationships can specify relationships among these conditions of different goals. For instance the sub-goals of the *Explore asteroid* goal are being achieved sequentially, which is expressed by necessary contributions between their post- and pre-conditions.

Goal decomposition diagrams explicitly model the problem decomposition and help to analyze the system requirements. They can be used either to goal-based requirements specification, means-ends analysis, or to requirements-driven application design [3] and identification of the responsibilities of particular entities within the system, see Sect. 3.3 for details.

3.3. Organization of swarms

As the class diagram in Fig. 4 shows, several kinds of spacecrafts are defined. A *Transport Ship* which as-

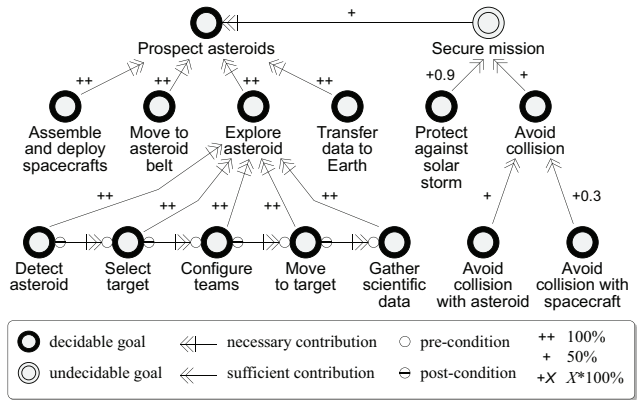


Figure 3. Top-level mission goals.

sembles and deploys the small spacecrafts, an abstract PAM *Spacecraft* which represents all kinds of small PAM spacecrafts, a *Leader* coordinating all activities within a sub-swarm, a *Messenger* facilitating communications among the workers, leaders, and mission control on Earth, and a *Worker* bearing an instrument and gathering scientific data. The concrete workers are of different kinds depending on what type of instrument they provide, e.g. magnetometer, x-ray, gamma-ray, visible/infrared, or neutral mass spectrometers.

In addition to the spacecraft classification hierarchy, the class diagram shows also the responsibilities of particular spacecraft kinds expressed in terms of possible goal commitments (modeled by means of mental properties). These goals (defined in Fig. 3) can be instantiated in certain situations during the operation of the PAM mission, what means that their owners are committed to achieve them. This mechanism represents a practical means for modeling explicit traceability relationships from a goal (problem) decomposition models to the system architecture models (used e.g. in the requirements-driven application design [3]).

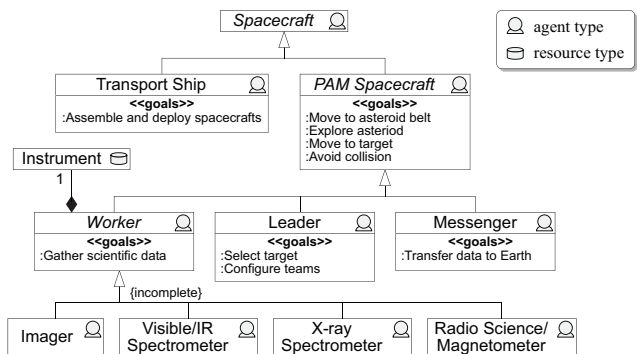


Figure 4. Types of spacecrafts/agents.

Internal organization of a swarm expressed in terms of organization units, entity roles and their social relationships is depicted in Fig. 5. Organization unit Swarm consists of several other organization units called Sub-Swarms. Each sub-swarm includes the role of a leader (Subsw Leader), the role of a sub-swarm messenger (Subsw Messenger), and a number of specialized Teams. Each Team, an organization unit itself, comprises one or more specialized Workers and the role of a Team Messenger. Subsw Leader is superordinate to the Subsw Messenger and Teams. Teams with each other, workers within a team, and teams with sub-swarm messengers cooperate on a peer-to-peer basis.

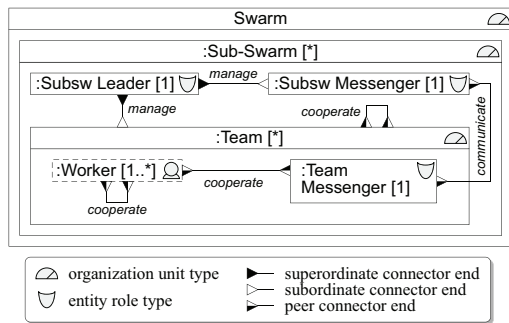


Figure 5. Social structure of swarm.

An example of the ability of an agent type to play a role of a particular type is illustrated in Fig. 6. In this model fragment a Messenger agent can play either a role of Subsw Messenger (sub-swarm messenger) or a Team Messenger. In an exceptional situation (described further in Sect. 4.1) an agent of the type Worker can play the role of a Team Messenger.

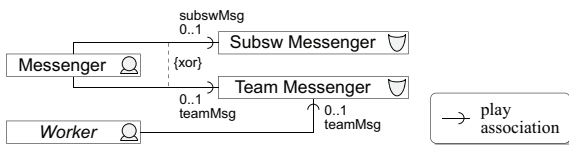


Figure 6. Playing roles by agents.

To optimize the data acquisition, PAM spacecrafts work together in ‘Virtual Instrument Teams’. Specification of several concrete types of teams and their constituents (agents which represent different workers) is depicted in Fig. 7. For simplicity we omitted team messengers from each type of team.

4. Autonomic behavior

This section provides examples of modeling scenarios used to realize autonomic behavior of the PAM system.

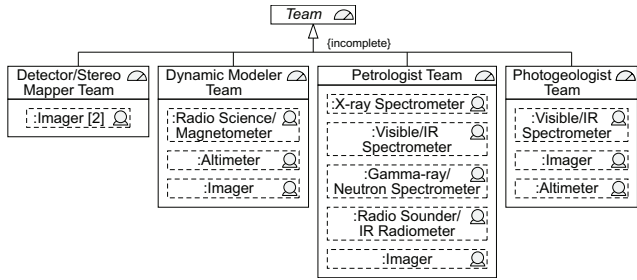


Figure 7. Virtual instrument teams.

4.1. Self-configuration

PAM must be fully (re-)configurable to support concurrent exploration and examination of hundreds of asteroids. Resources must be dynamically configured at both the swarm, sub-swarm, and team levels, in order to coordinate science operations while simultaneously maximizing resource utilization.

Appropriate teams of spacecraft are configured to conduct optimal science operations at the asteroids. When the science operations are completed, the team disperses for possible reconfiguration at another asteroid site. This configuring and reconfiguring continues throughout the life of the mission.

Reconfiguring may also be required as the result of a failure or anomaly of some sort. Fig. 8 shows the emergent activity applied when worker’s instrument is damaged and the worker can take the role of a team messenger. A worker notifies the damage of its instrument and sends a report to its sub-swarm leader. The leader then checks whether this worker can be used as a messenger in any of the sub-swarm teams (e.g. some team messenger is missing entirely or a worker is temporarily playing the role of a messenger). If not, the sub-swarm leader asks other sub-swarm leaders whether their teams need a messenger. If another sub-swarm needs a messenger, its leader returns a reassignment confirmation containing details about the reassignment, i.e. space navigation instructions, messenger software downloading instructions, etc. In the case if the damaged worker is going to play the role of a team messenger, its sub-swarm leader prepares the reassignment command, the damaged worker leaves the current team by stopping to play its role of a team worker, downloads the messenger software, navigates to its new team (if was reassigned to another team), and starts to play the role of a team messenger. In the case if no messenger reassignment is necessary, the sub-swarm leader registers its worker for possible future reassignment. The presented scenario is simplified and does not show e.g. how the worker’s sub-swarm leader solves the problem of a missing instrument, the negotiation of sub-swarm leaders about which sub-swarm obtains new team messenger, etc.

These behaviors can be modeled by other parallel activities.

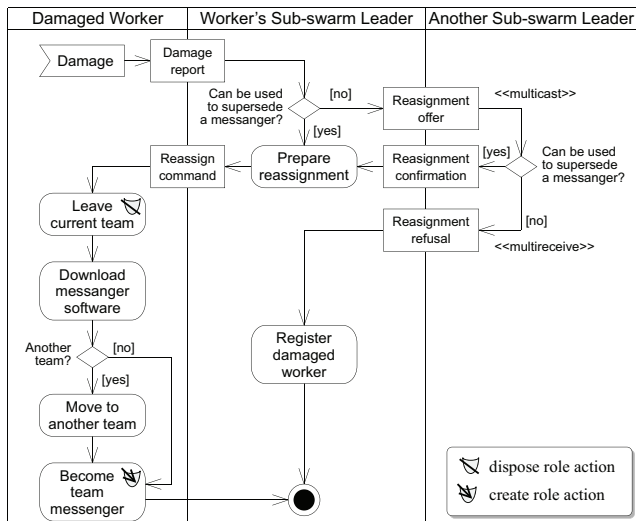


Figure 8. Failure of worker's instrument.

4.2. Self-optimization

Self-optimization is important to mission efficiency and flexibility. Leaders self-optimize primarily through learning and improving their ability to identify asteroids that will be of interest. Messengers self-optimize through positioning themselves appropriately. Workers self-optimize through learning and experience. Self-optimization at the system level propagates up from the self-optimization of individuals.

The interaction of a team messenger to optimize its position relatively to the team workers and sub-swarm messenger is depicted in Fig. 9. A team messenger requests all its team workers and a sub-swarm leader to provide their positions, and they in turn reply with the requested information. In the case a worker does not respond, that could mean it is damaged or lost, the team messenger informs the sub-swarm leader about it. Based on the obtained information, the team messenger adjust its own position to optimize the communication with all the concerned spacecrafts. This scenario is continuously executed in order to: (1) adjust the messenger's position to balance its communications, and (2) identify lost workers.

4.3. Self-healing

Spacecrafts, teams, sub-swarms, and whole swarm must heal themselves to recover from both mistakes and failures, including those caused by damage due either to solar storms or to a collision with an asteroid or another spacecraft.

An example of a team self-healing behavior has already been presented in Fig. 8. This scenario is a part of more

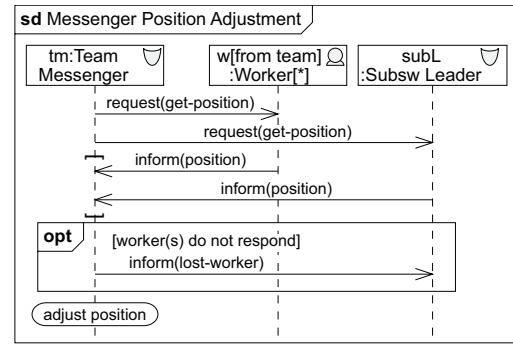


Figure 9. Messenger's position adjustment.

complex team self-healing behavior which enables to keep a team in operation even after some of its members are damaged or failed. In this particular example, a worker after appropriate self-configuration of its control software (being changed to the messenger software) can replace missing team messenger.

This example also shows overlapping of self-configuration and self-healing scenarios as discussed in [8].

4.4. Self-protecting

In addition to protection from collision with asteroids and other spacecraft, PAM teams must protect themselves from solar storms, where charged particles can degrade sensors and electronic components, and destroy solar sails. PAM spacecrafts must re-plan their trajectories, or, in worst-case scenarios, must go into "stand by" mode to protect their sails and instruments and other subsystems.

The interaction of spacecrafts used to protect them against a solar storm is depicted in Fig. 10. After receiving a message from the Earth control center or from a sub-swarm leader informing about a solar storm, a sub-swarm messenger broadcasts this information to the other sub-swarm messengers. All sub-swarm messengers then inform their leaders, the leaders inform their team messengers which in turn inform all team workers. Each spacecraft after receiving a warning message and performing necessary communication, puts itself to a "stand by" mode.

An analogous interaction is applied also when the sun storm is over and the spacecrafts should return to normal operational mode, but the "solar-storm-over" message is sent and the "wake up" status is achieved instead.

5. Conclusions

In this paper we have documented a selection of examples of how AML can be applied to the PAM application.

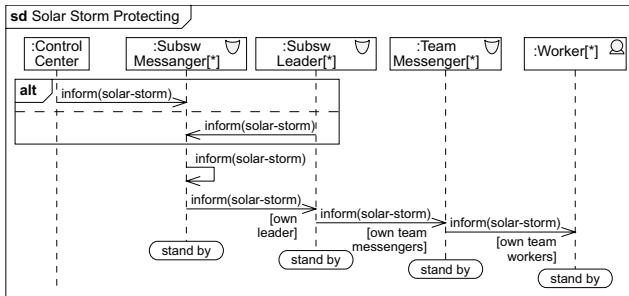


Figure 10. Solar storm protecting interaction.

Our intention has been to demonstrate how visual modeling of autonomic systems can help to improve the clarity, consistency and accuracy of designs, especially when faced with the intrinsic complexity of systems targeted for, or based on, the application of autonomic computing principles.

We believe that AML lends itself particularly well to modeling this type of system due to its specification as an extension to UML 2.0 dealing with the architectural and behavioral aspects of autonomous agent systems. Although software agents relate to only certain aspects of autonomic computing, many of the modeling principles remain consistent and as has been shown in this paper, can be quite effectively applied. AML not only helps to more clearly comprehend the structure and form of an autonomic application, it also allows systems to be modeled from different perspectives allowing designers greater flexibility in the way that they express ideas and concepts.

In order to provide a validation the modeled system, AML can also be transformed to formal specifications upon which formal validation techniques could be applied (see [5] for details). Another option is given by CASE tool support for AML allowing forward engineering to code. This latter feature in particular can be useful when creating simulation software for validating systems such as PAM before actual construction and deployment. CASE tool support for AML is available in the form of the Living Systems Technology Suite ¹. This suite is a professional development environment for producing applications based on software agent technology. AML is used by LS/TS as the means of designing applications whereby models can be transcribed directly into application code using a specialized modeling CASE tool.

Although far from a thorough demonstration of the scope of AML, we have attempted to provide didactic evidence that by virtue of its focus on modeling agent systems, it provides a powerful, comprehensive and easy-to-understand

¹Further information on The Living Systems Technology Suite (LS/TS) is available from <http://www.whitestein.com/pages/solutions/ls.ts.html>

tool for modeling autonomous and autonomic systems. AML has already been demonstrated, through commercial application, to be a powerful means of modeling systems containing autonomous software entities. We have now established that AML is also well suited to the modeling of autonomic systems and can intrinsically model aspects of self-* principles especially enacted through the behaviors of autonomous agents. In fact, modeling systems in this manner not only helps to clarify and simplify the analysis and design aspect of creating agent and autonomic systems, it also helps to identify where flaws and problems may lie in less well-formed and/or complex system designs.

References

- [1] R. Cervenka and I. Trencansky. Agent Modeling Language: Language specification. Version 0.9. Technical report, Whitestein Technologies, 2004.
- [2] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
- [3] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. Towards requirements-driven autonomic systems design. In *Proceedings of the 2005 workshop on Design and Evolution of Autonomic Application Software (DEAS)*, pages 1–7, May 2005.
- [4] M. Mamei and F. Zambonelli. Self-maintaining overlay data structures for autonomic distributed computing. In *Proceedings of the Second International Conference on Automatic Computing (ICAC'05)*, pages 376–377, 2005.
- [5] C. Rouff, W. Truszkowski, J. Rash, and M. Hinchey. Formal approaches to intelligent swarms. In *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03)*, page 51, 2003.
- [6] I. Trencansky and R. Cervenka. Agent Modelling Language (AML): A comprehensive approach to modelling MAS. *Informatica*, 29(4):391–400, 2005.
- [7] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. NASA's swarm missions: The challenge of building autonomous software. *IT Professional*, 06(5):47–52, September/October 2004.
- [8] W. Truszkowski, J. Rash, C. Rouff, and M. Hinchey. Asteroid Exploration with Autonomic Systems. In *Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04)*, page 484, 2004.
- [9] T. D. Wolf and T. Holvoet. Towards autonomic computing: Agent-based modelling, dynamical systems analysis, and decentralised control. In *Proceedings of the First International Workshop on Autonomic Computing Principles and Architectures (Tianfield, H. and Unland, R., eds.)*, pages 10–18, 2003.