Goal-Oriented Monitoring Adaptation : methodology and patterns

Antoine Toueir, Julien Broisin, Michelle Sibilla

Goal-Oriented Monitoring Adaptation: Methodology and Patterns

Antoine Toueir, Julien Broisin, and Michelle Sibilla

IRIT, University Toulouse III - Paul Sabatier
118 rue de Narbone, 31062 Toulouse, France
{toueir,broisin,sibilla}@irit.fr

Abstract. This paper argues that autonomic systems need to make their distributed monitoring adaptive in order to improve their "comprehensive" resulting quality; that means both the Quality of Service (QoS), and the Quality of Information (QoI). Thus, we propose a methodology to design monitoring adaptation based on high level objectives (*goals*) related to the management of quality requirements. One of the advantages of adopting a methodological approach, is that monitoring reconfiguration will be conducted through a consistent adaptation logic. Starting from a model-guided monitoring framework, we introduce our methodology to assist human administrators in eliciting the appropriate quality goals piloting the monitoring. Moreover, some monitoring adaptation patterns falling into reconfiguration *dimensions* are suggested and exploited in a cloud provider case-study illustrating the adaptation of Quality-Oriented monitoring.

Keywords: Quality requirements, adaptive monitoring, autonomic systems, goal-oriented adaptation.

1 Introduction

Autonomic systems that are implemented by virtue of their four characteristics self-configuration, self-optimization, self-healing and self-protection, are serving the main principle of making them self-managed to achieve high level objectives [1]. In practice, the four self-* characteristics are realized by implementing the MAPE-K (*Monitoring, Analyzing, Planning, Executing - Knowledge*) loop modules. This implementation is either embedded within a resource, or distributed over several resources. However, the monitoring module of MAPE-K loop plays a crucial role, since wrong decisions might be taken by the analyzing & planning modules, if they were provided with interrupted or wrong information. Therefore, autonomic systems need to ensure the quality of information (*e.g.,* correctness, freshness, timeliness, accuracy, etc.) exposed by the distributed monitoring modules.

Within autonomic systems, monitoring is usually quality-oriented. In other words, the underlying monitoring instruments metrics and evaluates them against quality specifications expressed via *Service Level Agreements* (SLAs) or management high level objectives. Since the management system could provide the

possibility to renegotiate or modify the QoS specification afterward, and also, various management needs could be distinguished during the management system lifetime, the monitoring system has to adapt its behavior according to these new requirements. To resume, the monitoring of autonomic systems has to be capable of configuring the underlying gathering mechanisms (*i.e.,* polling & notification) carrying the monitoring functions (*e.g.,* measuring, gathering, evaluating, filtering, etc.) starting from quality specification, as well as reconfiguring those mechanisms based on quality requirements.

Most of the time, reconfiguration is held through ad-hoc logic. But this approach isn't suitable for reuse in other scenarios, and also doesn't satisfy high level objectives; which are extended at the autonomic system whole scale. To overcome these issues, first, we adopted the *Requirements Engineering* methodology to design monitoring adaptation; it starts from high level goals, and ends up with the (re)configuration of monitoring mechanisms. However, reconfiguration questions such as: why to delay launching some monitoring mechanisms? Why to substitute remote agents? What determine the instrumentation of particular set of metrics but not another one? need to be answered. In other words, identifying goals representing the "starting point" for deriving monitoring (re)configuration is a big challenge. Thus, besides adaptation methodology, this paper answers these questions by proposing monitoring adaptation patterns to assist human administrators in designing meaningful adaptations, that increase the overall quality of autonomic systems.

The paper is organized as follows: Section 2 points out the weaknesses of other monitoring adaptation approaches; Section 3 gives an overview of the monitoring framework at the basis of our contributions; the methodology of designing goal-oriented adaptive monitoring is presented in Section 4; the reconfiguration dimensions as well as monitoring adaptation patterns are discussed in Section 5 and then applied on a case-study in Section 6; finally, we conclude by listing our perspectives in Section 7.

2 Related Work

This section enumerates some existing trends focusing on (i) adapting the QoS monitoring in autonomic systems [2,3,4,5,6], and (ii) designing patterns regarding the distributed deployment as well as the adaptation of MAPE loop modules [7,8,9].

Collecting additional metrics or joining managed resources is addressed in [2,3,5] either to adapt monitoring to meet SLA modifications, or to deal with the managed scope changes, or even to operate a "minimal" monitoring that is able to be extended in case of SLA violations. Indeed, the capability of scaling up/down the monitored metrics and resources is important as an adaptation action. But, it isn't clear whether this capability could be applied in other scenarios for distinct objectives, if so, how that could be feasible.

Runtime deployment of monitoring resources (*i.e.,* managers, probes) is discussed in [3,4,6] either to integrate monitoring into the SLA management lifecycle of large scale systems, or to replace failed managers, or even to monitor

metrics concerning particular paths or segments. But here also, besides the undeniable gains of deploying monitoring resources during runtime, we don't see how the system administrators can orchestrate the monitoring adaptation of the distributed modules among several collaborating managers. The orchestration isn't a trivial task, because a given quality objective may need to be extended on several managers and treated differently on each of them. Thus, a "simple deployment" of new monitoring resources isn't enough to realize orchestration.

Inspired from the autonomic computing reference architecture proposed in [7], patterns regarding the distribution of the MAPE loop modules were proposed in [8,9]. Those patterns are useful in term of design reuse as well as clarifying the application contexts and benefits, but they target mainly the deployment of the monitoring modules rather than the monitoring behavior itself. That is, once the autonomic system is designed and deployed based on the proposed patterns, the monitoring conserves its behavior. Consequently, the management system knowledge won't exceed a "maximum ceiling" and the management will be limited regarding treating new situations; as it has specific vision reflecting the same aspects all the time.

To resume, most of the studied work focuses either on the auto-configuration of monitoring, or on the reconfiguration of the functional system based on the knowledge produced from the QoS monitoring, or even the reconfiguration of the monitoring itself based on static purposes. Contrary to the studied work, we argue that increasing the QoS of the whole system begins from designing adaptive monitoring, that is derived from and satisfies high level quality requirements, and interpreted by several managers.

3 The Enriched Adaptive Monitoring Framework

Our approach is based on a 3-layered framework [10,11,12] illustrated in Figure 1, and defines three capabilities required to control monitoring: being configurable, adaptable and governable. This framework operates monitoring mechanisms without any consideration regarding agents or management protocols.

The **configurability layer** stands on the Distributed Management Task Force (DMTF) Common Information Model (CIM) standard. This low level layer aims at representing, in addition to the managed resources, the metrics [13] and their gathering mechanisms as well [10]. Moreover, those models have been enriched with the concept of *Monitoring Mode*. The latter encapsulates: metrics (*Aspects of Interest* to be instrumented), constraints (thresholds to be checked once associated metrics are instrumented), indications (notifications to be raised once metrics are violated), as well as two types of subscriptions to deliver both metrics values and raised indications to the appropriate destinations (see Figure 1). The **adaptability layer** provides an interface encapsulating operations to be applied on the lower layer models. Those operations constitute a "control interface" to update the attributes of both monitoring modes and metrics, thus the underlying gathering mechanisms will be reconfigured, and consequently the monitoring adapts its behavior. Finally, the **governability layer** is the top

level layer representing the "intelligence" of the monitoring adaptation. To express the quality requirements, it uses Event/Condition/Action (ECA) policies to describe *when* and *how* adaptation should take place, that is, they determine the situations during which the adaptability layer operations should be invoked.

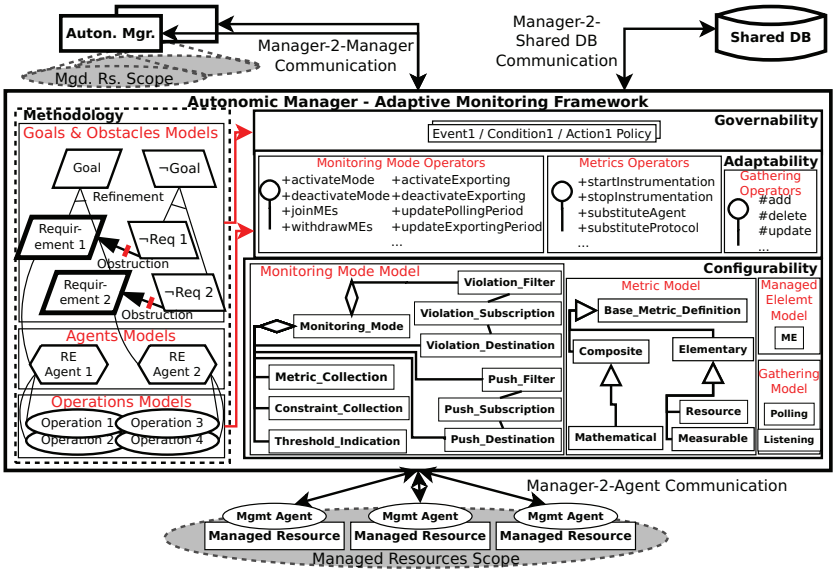


Fig. 1. Adaptation Methodology & Enriched Monitoring Framework

4 A Goal-Oriented Methodology for Adaptive Quality-Oriented Monitoring

Thinking that one of the existing software engineering approaches should answer our needs, we looked forward a suitable method for designing monitoring adaptation. The origin of Requirements Engineering (RE) goes back to the need to avoid crucial mistakes committed at the project design phase, and aims at building systems solving real-world problems. This methodology applies iterative activities about "*eliciting, evaluating, documenting, consolidating and changing the objectives, functionalities, assumptions qualities and constraints that the system-to-be should meet based on the opportunities and capabilities provided by new technologies*" [14]. Among multiple RE approaches, **K**eep **A**ll **O**bjectives **S**atisfied (KAOS) is adopted as RE goal-oriented method, due to its formal assertion layer that proves correctness and completeness of goals [15].

In KAOS the system-to-be is divided into various models. The **goal models** elicit goals representing high level behavioral prescriptions of the system-to-be (the monitoring system in this study). Each goal may serve one or more objectives, and a given goal is realized through the cooperation of several components or actors,

the so-called *Agents*[1]. Goals are decomposed into sub-goals via a refinement process (see Section 6), where the most refined goals are called *Requirements*, or *Leaf Goals*; in KAOS, each of those leaf goals is assigned to a specific agent in order to be realized. Others goals, depicted within the **obstacle models**, are deduced from the goal models and prevent the satisfaction of the latter. In our study, since goals converge on the quality of monitoring, their obstacles will be related to the monitoring failures. The **operation models** are composed of the sets of internal operations carried by agents in order to realize the *Leaf Goals*. Finally, the **object models** identify the system-to-be objects.

By iterating refinement process on goals and obstacles, leaf goals will be identified. Once leaf goals are determined, both of ECA policies (to be inserted into the governability layer) and agents (invoking operations of the adaptability layer) will be reconnoitered. Thus, monitoring adaptation is automatically handled, and high level goals remain satisfied. However, human administrators have to refine manually the high level objectives they want to reach, in order to identify the leaf goals. To facilitate this task, we investigated the monitoring aspects that are subject to adaptation. As a result, we have identified various leaf goals belonging to four *dimensions* (*i.e.,* Spatial, Metric, Temporal, Exchange) [16]. In Section 5, we pursuit in proposing monitoring adaptation patterns falling into those dimensions, in order to assist human administrators in refining goals.

5 Dimensions and Patterns

With regard to refinement process, besides the basic AND/OR-decompositions, we rely on some predetermined correct and complete refinement patterns proved mathematically [17]. Those patterns refine *Achieve* goals of the form $P \Rightarrow \Diamond Q$ (see Table 1), and written in *Linear Temporal Logic* (LTL) classical operators where $\Diamond$, $\Box$ and $\mathcal{W}$ mean respectively *some time in the future*, *always in the future*, and *always in the future unless*. Starting from a given goal (P), *milestone pattern* identifies one or many intermediate goals ($R, [...]$) that must be reached orderly before reaching the ultimate one (Q). Rather, *case pattern* identifies the set of different and complete cases ($P1, P2$) for reaching final goals ($Q1, Q2$) that OR-decompose the ultimate goal (Q). Finally, the *guard pattern* requires the recognition of a condition (R) before achieving the ultimate goal (Q).

In order to clarify the exploitation contexts, pattern goals and requirements, as well as the various situations in which they may apply to, our pattern structure encompasses: context, pattern refinement, and examples. Notice that we are focusing on adaptation actions taken at the autonomic manager side only. Thus, investigating adaptations at the agent side is out of scope. In addition, the patterns are refined using KAOS graphical language [14].

[1] Notice that *Agents* in networks and systems management are entities responding to management requests coming from other management entities called *Managers*; therefore the term "*Agent*" in RE has a different meaning.

Table 1. Patterns Refining *Achieve* Goals ($P \Rightarrow \Diamond Q$) [17]

Pattern	**Subgoal 1**	**Subgoal 2**	**Subgoal 3**
Milestone Pattern	$P \Rightarrow \Diamond R$	$R \Rightarrow \Diamond Q$	
Case Pattern	$P \wedge P1 \Rightarrow \Diamond Q1$	$P \wedge P2 \Rightarrow \Diamond Q2$	$\Box(P1 \vee P2)\ Q1 \vee Q2 \Rightarrow Q$
Guard Pattern	$P \wedge \neg R \Rightarrow \Diamond R$	$P \wedge R \Rightarrow \Diamond Q$	$P \Rightarrow P\ \mathcal{W}\ Q$

5.1 Exchange Dimension Pattern

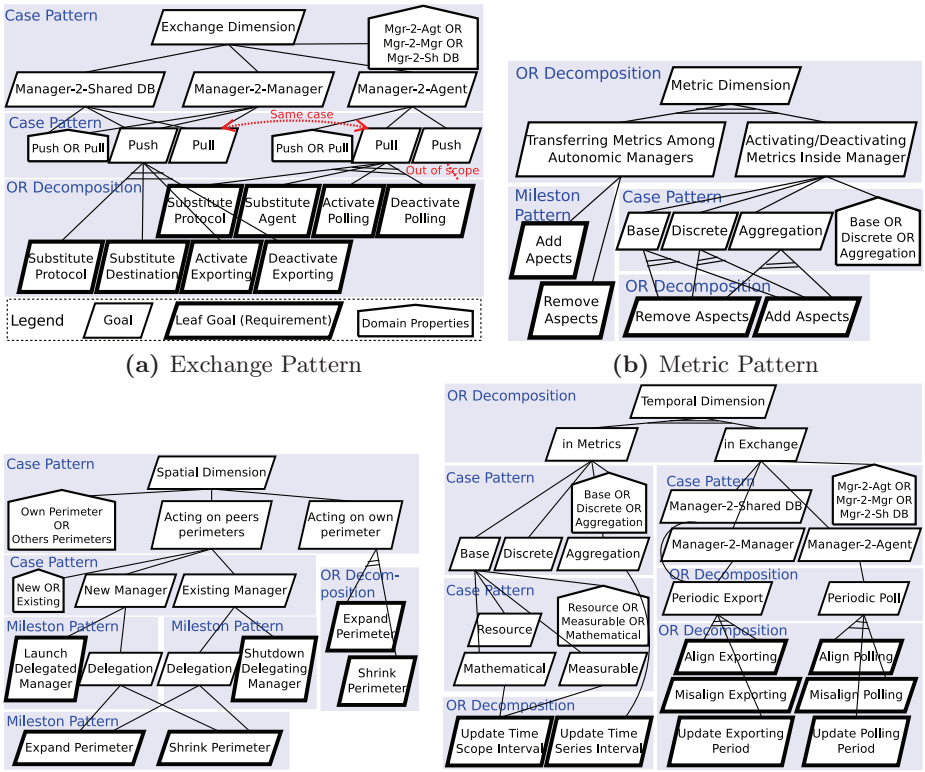
Context. Relying on IBM blueprint reference architecture [7], autonomic systems could distribute the MAPE loop over multiple collaborating autonomic managers. Each of which is responsible for managing particular scope of managed resources. Patterns belonging to this dimension are useful to overcome metrics gathering and delivering problems. Those problems could affect either metrics values, communication reliability between the information sources and destinations, or even on trustworthiness of those sources and destinations.

Pattern Refinement. Communications inside autonomic system could be classified according to the entities involved in information exchange (*i.e.,* managers, agents, shared databases). Therefore, we identify three communication classes: Manager-2-Agent, Manager-2-Manager, and Manager-2-Shared Database (see Figure 1). By taking into consideration push and pull modes[2], along with previous communications classes, we use *case pattern* for the first two refinement levels to cover all possible cases. Based on the triplet $\langle$ *Information Source, Communication Protocol, Information Destination* $\rangle$, the Manager-2-Agent pull mode will be OR-decomposed into *Substitute Agent* and *Substitute Protocol* leaf goals. Rather, *Substitute Protocol* and *Substitute Destination* OR-decompose both Manager-2-(Manager/Shared DB) push mode. Besides, *Activate/Deactivate Polling & Exporting* leaf goals are elicited to launch and stop polling & exporting (see Figure 2a).

Since a manager responds to pull requests in both Manager-2-(Manager/ Shared DB) pull mode communications, it is considered as agent (because it is the source of information); therefore, this case becomes identical to Manager-2-Agent pull mode. Moreover, adaptation actions related to Manager-2-Agent push mode are not treated because they need to be held at the agent side.

Examples. This pattern is suitable for the following cases: (1) Increasing accuracy or precision of pulled/pushed metrics values, by replacing information source. (2) Querying more available agents, or blocking fake agents trying to integrate the distributed management system. (3) Securing the communication between information sources and destinations. (4) Modifying information destination when changing the topology of collaborating autonomic managers.

[2] In *pull*, entity needing information solicits the one possessing it, that responds with queried information; while in *push*, entity possessing information reports it to others.



(a) Exchange Pattern **(b)** Metric Pattern

(c) Spatial Pattern **(d)** Temporal Pattern

Fig. 2. Dimensional Patterns

5.2 Metric Dimension Pattern

Context. The main idea behind building autonomic systems is to delegate decisions, that human administrators are used to make, to the autonomic systems themselves. Thus, to be able to make "wise" decisions, monitoring system needs to instrument specific metrics that could be activated/deactivated according to the management needs during runtime. Patterns belonging to this dimension are useful to control the trade-off between constructing richer knowledge and monitoring the information that is necessary for management.

Pattern Refinement. Metric instrumentation must be thought at the whole management system level. In other words, a given autonomic manager could activate/deactivate instrumentation of particular metrics, but when deactivating metrics on that manager, it doesn't mean necessarily that those metrics are "abandoned", because they could be transferred to other collaborating autonomic manager on which they are activated. These two cases are OR-decomposing the first refinement level. Regarding metrics manipulation inside autonomic

manager, the second refinement level uses *case pattern* to cover metric classes. Our research exploits both CIM Metric Model [18] classifying metrics into *Base, Discrete & Aggregation*, as well as our mathematical extension [13] classifying base metrics into *Resource, Measurable & Mathematical*. Each of these classes is OR-decomposed using *Add Aspects* and *Remove Aspects* leaf goals. On the other hand, the transfer of metrics among autonomic managers could be refined through *milestone pattern*, when metrics are activated on the collaborating manager first (*Add Aspects* in Figure 2b, as Subgoal 1 in Table 1), and then removed from the delegating one (*Remove Aspects*, as Subgoal 2).

It is worthy to precise that previously mentioned *Aspects* are representing "metric definitions", rather than "metric values". The former encompasses attributes related to the nature of metric (*e.g.,* data type, unit), while the latter describes the instrumented values and their relevant contexts.

Examples. This pattern can be applied in the following cases: (1) Troubleshooting, or applying root cause analysis algorithms. (2) Modifying the hierarchical topology of the management system by instrumenting aggregated metrics to be exported to other managers or shared DBs. (3) "Engineering" the distribution of monitored metrics among autonomic managers.

5.3 Spatial Dimension Pattern

Context. As mentioned earlier, in autonomic system, each manager is responsible for managing a set of managed resources. In many cases, the number of users consuming the autonomic system services may oscillate rapidly, or even become quite important in term of size. Thus, managed resources are subject to be joined/withdrawn during runtime. Patterns belonging to this dimension are useful to react to the important changes of the managed resources scope.

Pattern Refinement. As management of autonomic systems is orchestrated by the collaboration of multiple autonomic managers, each of which can act on its own perimeter, as well as the perimeters of its collaborating peers. Thus, the first refinement level uses *case pattern* to cover these two cases. In fact, acting on its own perimeter is OR-decomposed using *Expand* and *Shrink Monitoring Perimeter* leaf goals to join/withdraw resources respectively into its managed scope. Rather, acting on others perimeters is refined using *case pattern* into deploying a new manager, or soliciting an existing one. First, the case of deploying a new manager is refined using *milestone pattern* into launching manager (*Launch Delegated Manager* in Figure 2c, as Subgoal 1 in Table 1), and then, delegating perimeter (*Delegation*, as Subgoal 2). In its turn, delegation goal is also refined through *milestone pattern* into joining delegated perimeter on the delegated manager (*Expand Perimeter*, as Subgoal 1), and then, deleting this perimeter from the delegating manager (*Shrink Perimeter*, as Subgoal 2). About the second case, where acting is held on existing manager, it is refined twice, using *milestone pattern*, into delegating the whole perimeter to the delegated manager (*Delegation*, as Subgoal 1), and then shutting down the delegating one (*Shutdown Delegating Manager*, as Subgoal 2).

Examples. This pattern is suitable for the following cases: (1) Load balancing of monitoring among autonomic managers. (2) Supporting scalability of the autonomic systems. (3) Decreasing the number of monitoring resources.

5.4 Temporal Dimension Pattern

Context. Temporal aspects are decisive factors in adapting monitoring behavior. Notice that the previous patterns are explained without considering time notions, but in fact, they imply some temporal aspects. Patterns belonging to this dimension are useful to overcome, among others, both temporal violations and scheduling problems, as well as to tune the analysis over the instrumented metrics. However, these two cases are far from being exhaustive, and time intervenes in a lot of other cases.

Pattern Refinement. Regarding information exchange, once again, we use *case pattern* to represent the same cases identified in *exchange* dimension. Obviously, dealing with information exchange temporal aspects, means that exchange is done iteratively and not once. Thus, Manager-2-Agent case is OR-decomposed into periodic poll, and both Manager-2-(Manager/Shared DB) cases are OR-decomposed into periodic export. We distinguish two levels of temporal granularity: the fine-grained level deals with an individual polling (exporting), whereas the coarse-grained level addresses a collective polling (exporting). Based on this distinction, we identify various leaf goals OR-decomposing periodic poll (export), where: *Update Polling (Exporting) Period* to update the frequency of a given polling (exporting), *Align Polling (Exporting)* to launch a set of synchronized parallel pollings (exported metrics) at the same time, and *Misalign Polling (Exporting)* to launch pollings (exported metrics) according to a "relative offset" delaying their launching moments one another (see Figure 2d).

Regarding metrics calculation, we identify the case of modifying the temporal interval covered by the metric value[3]. Therefore, *case pattern* is used twice to cover all possible metric classes previously mentioned. But, we refine only measurable, mathematical & aggregation metrics, because time has a sense in their calculation, but not the others. Thus, we OR-decompose measurable & mathematical metrics using *Update Time Scope Interval*, while *Update Time Series Interval* OR-decomposes aggregation metrics (see Figure 2d).

Examples. This pattern is suitable for the following cases: (1) Controlling (*e.g.,* relaxing, stressing) the monitoring load on autonomic managers, network paths among autonomic managers and shared DBs, as well as remote agents. (2) Tuning temporal parameters of metrics analysis.

6 Case-Study

Context. Our scenario rolls in a cloud data center providing to the virtual machines (VMs) owners a continuous monitoring of their enforced SLAs metrics.

[3] For instance, the *throughput* is not an instantaneous metric, and the validity of its value equals to the temporal interval through which that value was measured.

Each VM integrates two agents (primary: *MIB-II SNMP* & secondary: *SBLIM ProviderCmpiBase*) providing metrics reflecting the performance level of that VM. In most large scale systems, distributed agents push metrics periodically; in our case, agents push metrics each 10 seconds to specific pre-configured autonomic managers. We assume that our studied SLA template distinguishes two time-slots: metrics must be refreshed at the client side with a freshness falling into the range of [3-6] seconds during the first time-slot, and a range of [30-40] seconds for the second one. The SLAs metrics values are instrumented and delivered automatically through polling and exporting respectively in a manner that, once new SLA is enforced, the autonomic managers pull metrics with the lowest freshness value (3 seconds).

Objectives. Based on the data center management strategies, human administrators identify three high level monitoring goals: *Respect Metrics Freshness* makes sure that SLAs are monitored appropriately regarding freshness, *Minimize Monitoring Cost* aims at limiting the resources dedicated to monitoring as much as possible, and *React to Gathering Problems* operates resilient gathering mechanisms after analyzing the potential reasons of gathering problems.

Patterns. Several patterns could be exploited to refine the first objective. During the first time-slot, we use the *temporal pattern* to relax polling & exporting by updating their periods (*Update Polling & Exporting Period* in Figure 3) with respect to the highest freshness range (6 seconds). If delivering freshness violates the highest freshness, that would be a result of overloading manager [16], thus we apply the *spatial pattern* as a second alternative, and consequently, a new autonomic manager will be deployed to assist the overloaded one (*Launch Delegated Manager, Expand Perimeter & Shrink Perimeter*). As a third alternative, and in case that the overloaded autonomic manager monitors non-SLAs metrics (*i.e.,* physical servers healthiness), the *metric pattern* could be applied to transfer them to other manager, in order to relax the first one (*Add & Remove Aspects*). Since the second time-slot freshness ([30-40] seconds) is greater than agents push period (10 seconds), there is no need to poll metrics, nor to export all received metrics. Rather, we apply the *temporal pattern* to update the exporting period from [3-4] to [30-40] seconds (*Update Exporting Period*), and consequently, the first time-slot pollings will be stopped (*Deactivate Polling*).

The second objective is refined using *spatial pattern* in order to shutdown recently deployed managers, during the first time-slot. Thus, an underloaded manager delegates its whole perimeter to another one, and shutdowns itself (*Expand Perimeter, Shrink Perimeter & Shutdown Delegating Manager*). During the second time-slot, autonomic managers already deliver to clients around one-third of the metrics pushed by agents, thus no adaption actions are to be taken in regard with minimizing monitoring resources.

As the third objective deals with gathering problems (*e.g.,* lack of collected information), autonomic managers would act on the *exchange pattern*. Notably, they substitute either remote agent or communication protocol (*Substitute Agent & Protocol* in Figure 3). But, if "failures" at the agent side cause information

lack, in such case substituting protocol won't solve the problem. Therefore, we act on the *metric pattern* by launching troubleshooting to acquire more knowledge (*Add Aspects*) required to determine the appropriate substitution.

For all previous objectives, autonomic managers adapt their monitoring if they recognize adaptation stimuli. Therefore, we exploit *guard pattern* to apply adaptation actions (*Adaptation* in Figure 3, as Subgoal 2 in Table 1) as response to specific stimulus (*Guard*, as Subgoal 1), while maintaining the current monitoring behavior unless adaptation takes place (*Unless*, as Subgoal 3).

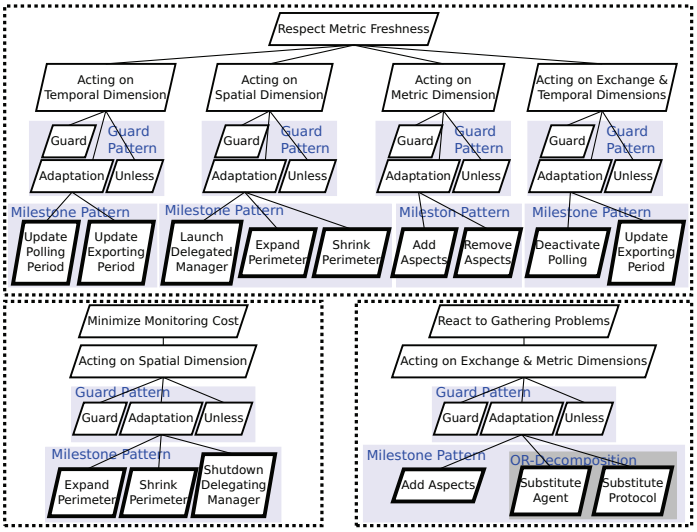


Fig. 3. Case-Study High Level Objectives Refinement

Applying Leaf Goals. In our case-study, autonomic managers are CIM servers operating model-guided monitoring. Moreover, the healthiness indicators of the physical servers, as well as each SLA template, are converted into a monitoring mode each, whereby metrics and constraints are encapsulated.

The *Update Polling/Exporting Period* leaf goals are realized through applying the corresponding methods on all SLA monitoring mode instances to stretch the time interval of collecting/delivering metrics.

The *Expand Perimeter* leaf goal is realized in two manners: first, in case of delegating managed resources to another autonomic manager, whereby the same SLA templates are already enforced (*i.e.,* SLAs monitoring modes are already instantiated), the delegated manager will **join** the yet transfered managed resources in their SLA monitoring mode instances. Otherwise the delegated manager will **activate** the appropriate monitoring mode instances over the transfered resources. However, the *Shrink Perimeter* leaf goal is realized through withdrawing the determined managed resources from their SLA monitoring modes.

Each metric definition instance belonging to a SLA monitoring mode is systematically associated with a *listening* instance to update that metric with the pushed values. But, the same metric definition is also associated with a polling instance to pull its values according to the SLA period. However, *Deactivate Polling* leaf goal is applied on all SLA monitoring mode instances to stop polling and consequently *listening* continues to instrument associated metrics with the agents push period.

Both *Add/Remove Aspects* leaf goals are realized respectively through monitoring mode activation/deactivation. Therefore, to transfer the servers healthiness indicators, the corresponding monitoring mode will be activated on the delegated manager and deactivated from the delegating one. Rather, autonomic manager activates the "gathering troubleshooting" mode to determine whether it must substitute agent or protocol. This mode encapsulates metrics collected from remote agents SNMPv2-MIB variables, such as: *snmpInBadVersions & snmpInBadCommunityNames*[4]. If the SNMP messages querying agent don't increment the "gathering troubleshooting" mode metrics of that agent, it means that the synthesized SNMP messages are correct, and probably the problem comes from the agent itself; in that case, *Substitute Agent* is applied.

7 Conclusion and Perspectives

Based on the Requirements Engineering, we proposed a goal-oriented approach for designing self-managed monitoring in autonomic systems. This approach assists human administrators to adapt the monitoring system behavior regarding quality requirements. We designed four reusable monitoring adaptation patterns falling into reconfiguration dimensions.

About the perspectives, and in order to validate our approach, we need to consider two validation levels. First, the completeness and correctness of the patterns refinement must be validated. Using proof theory, we can avoid missing some necessary requirements, and also, we can discover the available alternatives to refine a given goal [17]. On the other hand, once the monitoring system adaptation is modeled as a *transition system* (*i.e.,* set of states and transitions) through the *Temporal Logic*, we need to determine the critical properties to be checked (*i.e.,* invariance, safety, eventuality, fairness, and precedence) [19]. Using model checking, the properties satisfaction is checked, either on particular state, or path, or even the whole system model. Apart from validation, and besides enriching patterns, adaptation actions at the agent side need to be investigated, and orchestrated with those applied at the autonomic manager side.

[4] These variables describe the number of SNMP messages delivered to a SNMP agent with unsupported version and unknown "community string", respectively .

References

1. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41–50 (2003)
2. Grefen, P., Aberer, K., Ludwig, H., Hoffner, Y.: Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. International Journal of Computer Systems Science & Engineering 15, 277–290 (2000)
3. Roxburgh, D., Spaven, D., Gallen, C.: Monitoring as an sla-oriented consumable service for saas assurance: A prototype. In: 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 925–939 (2011)
4. Thongtra, P., Aagesen, F.: An adaptable capability monitoring system. In: 2010 Sixth International Conference on Networking and Services (ICNS), pp. 73–80 (2010)
5. Munawar, M.A., Reidemeister, T., Jiang, M., George, A., Ward, P.A.S.: Adaptive monitoring with dynamic differential tracing-based diagnosis. In: De Turck, F., Kellerer, W., Kormentzas, G. (eds.) DSOM 2008. LNCS, vol. 5273, pp. 162–175. Springer, Heidelberg (2008)
6. Nobre, J.C., Granville, L.Z., Clemm, A., Prieto, A.G.: Decentralized detection of sla violations using p2p technology. In: Proceedings of the 8th International Conference on Network and Service Management, pp. 100–107. International Federation for Information Processing (2012)
7. IBM Corp.: An architectural blueprint for autonomic computing. IBM White Paper (June 2005)
8. Weyns, D., et al.: On patterns for decentralized control in self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Self-Adaptive Systems. LNCS, vol. 7475, pp. 76–107. Springer, Heidelberg (2013)
9. Ramirez, A.J., Cheng, B.H.C.: Design patterns for developing dynamically adaptive systems. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010, pp. 49–58 (2010)
10. Moui, A., Desprats, T., Lavinal, E., Sibilla, M.: A cim-based framework to manage monitoring adaptability. In: 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM), pp. 261–265 (2012)
11. Moui, A., Desprats, T., Lavinal, E., Sibilla, M.: Information models for managing monitoring adaptation enforcement. In: International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE), Nice, July 22-27, pp. 44–50 (2012)
12. Moui, A., Desprats, T., Lavinal, E., Sibilla, M.: Managing polling adaptability in a cim/wbem infrastructure. In: 2010 4th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), pp. 1–6 (2010)
13. Toueir, A., Broisin, J., Sibilla, M.: Toward configurable performance monitoring: Introduction to mathematical support for metric representation and instrumentation of the cim metric model. In: 2011 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), pp. 1–6 (2011)
14. Van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley (2009)
15. Van Lamsweerde, A.: Requirements engineering in the year 00: a research perspective. In: Proceedings of the 2000 International Conference on Software Engineering, pp. 5–19 (2000)

16. Toueir, A., Broisin, J., Sibilla, M.: A goal-oriented approach for adaptive sla monitoring: a cloud provider case study. In: LATINCLOUD 2013, Maceió, Brazil (December 2013)
17. Darimont, R., Van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. In: Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering - SIGSOFT 1996, pp. 179–190 (1996)
18. DMTF Applications Working Group: Base metrics profile (December 2009)
19. Goranko, V.: Temporal logics for specification and verification. In: Proceedings of the European Summer School in Logic, Language and Information, ESSLI 2009 (2009)