Reconfiguration Patterns for Goal-Oriented Monitoring Adaptation

Antoine Toueir, Julien Broisin, Michelle Sibilla
IRIT, Université Paul Sabatier
Toulouse, France
Email: {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). In a previous work, we proposed a methodology to design monitoring adaptation based on high level objectives 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. However, eliciting the appropriate quality goals remains an area to be investigated. In this paper, we tackle this issue by proposing some monitoring adaptation patterns falling into reconfiguration *dimensions*. Those patterns aim at facilitating the adaptation design of monitoring behavior of the whole set of distributed monitoring modules part of autonomic systems. The utility of those patterns is illustrated through a case-study dealing with monitoring adaptation based on high level quality objectives.**

Keywords–Quality requirements; adaptive monitoring; autonomic systems; goal-oriented adaptation.

I. Introduction

Autonomic systems that are implemented by virtue of their four characteristics self-configuration, self-optimization, self-healing and self-protection, are serving the ultimate objective of making them self-managed to achieve high level objectives [1]. These objectives are strongly related to the quality level provided by autonomic systems. When large and complex systems are targeted, the self-management characteristic (self-*) is a key issue to deal with. Basically, self-management is thought as the auto-adaptation capability that brings the system to reach an absolute or preferable state. Concretely, the four self-* characteristics are realized by implementing the *Monitoring, Analyzing, Planning, Executing - Knowledge* (MAPE-K) loop modules. This implementation is either embedded within a resource, or distributed over several resources.

In MAPE-K loop, the monitoring module plays a crucial role, since wrong decisions might be taken by the analyzing & planning modules. Therefore, autonomic systems need to ensure the quality of information (*e.g.,* correctness, freshness, timeliness, accuracy, etc.) exposed by the distributed monitoring modules. Moreover, within autonomic systems, monitoring is usually QoS-oriented. Thus, the services implemented by the functional system must respect the required QoS level that is determined through the *Service Level Agreements* (SLAs) that have been agreed with clients. Since the management system (managing the functional system) could provide the possibility to renegotiate or modify the QoS specification afterward, the monitoring system has to adapt its behavior according to these new requirements and constraints.

To summarize, the monitoring of autonomic systems has to be capable of configuring the underlying mechanisms carrying the monitoring functions (*e.g.,* measuring, gathering, calculating, evaluating, filtering, etc.) starting from QoS specification, as well as reconfiguring those mechanisms based on quality requirements.

Most of the time, reconfiguration is held through ad-hoc logic (proposing solutions for particular scenarios dealing with specific issues). But, this approach is not suitable for reuse in other scenarios, and it also does not satisfy high level objectives extended over the whole scale of the autonomic system. To overcome these issues, we adopted a Requirements Engineering methodology to design monitoring adaptation; it starts from high level goals, and ends up with the configuration of monitoring mechanisms [2].

Right now, the key question is: how to identify goals representing the "starting point" for deriving monitoring (re)configuration? In other words, reconfiguration questions such as: why to delay launching some monitoring mechanisms? Why to substitute remote agents? How to aggregate alarms? What determines the monitoring of this set of metrics and not another one? Why to exchange metrics among distributed management entities? This paper deals with these questions by proposing monitoring adaptation patterns that assist human administrators in designing meaningful adaptations and thus increase the overall quality of the autonomic systems.

The work presented here relies on both a 3-layered adaptive monitoring framework [3][4][5] and our goal-oriented adaptation methodology [2]. We pursue this research by focusing on adaptation patterns dedicated to the identification of high level goals, together with their refinement. The paper is organized as follows: the next section gives an overview of the studied monitoring framework; the monitoring adaptation patterns are discussed in Section III; and then applied to a case-study in Section IV; before concluding, Section V enumerates other monitoring adaptation approaches and points out their weaknesses.

II. The Studied Adaptive Monitoring Framework

Our approach is based on a 3-layered framework [3][4][5] illustrated in Figure 1, and defines three fundamental capabilities required to control monitoring: being **configurable**, **adaptable** and **governable**.

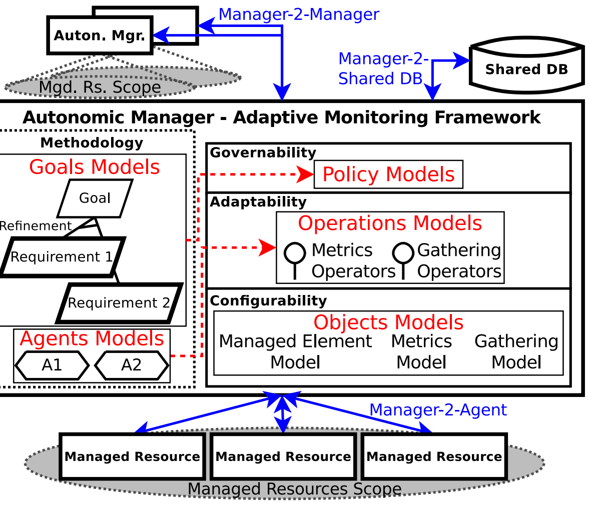

Figure 1: Adaptation Methodology & Monitoring Framework

The **configurability layer** relies on the Distributed Management Task Force (DMTF) Common Information Model (CIM) standard. In addition to the managed resources, this low level layer aims at representing the metrics [6] and the gathering mechanisms [3] that are required to monitor both the QoS provided by the functional system and the QoI of the monitoring system itself; this layer deals with both mechanisms. The **adaptability layer** provides an interface encapsulating operations to be applied on the lower layer models. Thus, the behavior of the underlying monitoring mechanisms will be reconfigured during runtime by invoking these operations. 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, in which contexts those operations of the adaptability layer should be invoked.

We exploit the Requirements Engineering (RE) to propose monitoring adaptation methodology, and to build configurability and adaptability models [2]. RE iterates activities of "*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*" [7]. Keep All Objectives Satisfied (KAOS) is adopted as RE goal-oriented method, due to its formal assertion layer that proves correctness and completeness of goals [8]. In KAOS, the system-to-be is divided into various models. *Goals Model* determines the objectives to be realized through that system (*e.g.,* minimizing monitoring cost). *Agents Model* comprises the agents (*e.g.,* human automated component) responsible for realizing the refined elicited goals. Notice that the term *Agents* in networks and systems management represents entities responding to management requests coming from other management entities called *Managers*; therefore, the term *Agent* in RE has a different meaning. *Operations Model* deals with the internal operations to be carried by agents (*e.g.,* updating polling period). *Object Model* identifies the system-to-be objects (*e.g.,* entities, agent, relationships).

Therefore, based on KAOS, our methodology identifies the

TABLE I: Patterns Refining Achieve Goals ($P \Rightarrow \Diamond Q$)

Pattern	Subgoal 1	Subgoal 2	Subgoal 3
Milestone	$P \Rightarrow \Diamond R$	$R \Rightarrow \Diamond Q$	
Case	$P \land P1 \Rightarrow \Diamond Q1$	$P \land P2 \Rightarrow \Diamond Q2$	$\Box(P1 \lor P2)$ $Q1 \lor Q2 \Rightarrow Q$
Guard	$P \land \neg R \Rightarrow \Diamond R$	$P \land R \Rightarrow \Diamond Q$	$P \Rightarrow PWQ$

high level quality objectives the monitoring framework carries on. By iterating a refinement process, we finally identify what it is called *leaf goals* or *requirements* (see Figure 1). Once the leaf goals are determined, both policies (to be inserted into the governability layer) and agents (invoking operations of the adaptability layer) will be recognized. Thus, monitoring system adaptation is automatically handled. However, human administrators have to manually identify the leaf goals according to the high level objectives they want to reach. To facilitate this task, we conducted an investigation about the monitoring aspects that could be subject to adaptation. As a result, we have identified various leaf goals belonging to four *dimensions* (*i.e.,* Spatial, Metric, Temporal, Exchange) [2]. Hereafter, in the next section, we propose monitoring adaptation patterns falling into those dimensions.

III. Monitoring Adaptation Patterns

With regard to the refinement process, besides the basic AND/OR-decompositions, we rely on some predetermined correct and complete refinement patterns proved mathematically [9]. Those patterns refine *Achieve* goals of the form $P \Rightarrow \Diamond Q$ (see Table I), and are written in *Linear Temporal Logic* (LTL) classical operators where $\Diamond$, $\Box$ and $\mathcal{W}$ mean *some time in the future*, *always in the future*, and *always in the future unless*, respectively. Starting from a given goal (P), *milestone pattern* identifies one (various) intermediate goal(s) (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 specific condition (R) before achieving the ultimate goal (Q).

In order to clarify the exploitation contexts, pattern goals and requirements, as well as some application situations, 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 [7].

A. Exchange Dimension Pattern

Context. Relying on IBM blueprint reference architecture [10], autonomic systems could distribute self-management (MAPE) loops over multiple collaborating autonomic managers. Each of them is responsible for managing a particular scope of managed resources. Patterns belonging to this dimension are useful to overcome metrics gathering/delivering problems. Those problems could manifest either on metrics values, reliability of communication between information sources & destinations, or even on their trustworthiness.

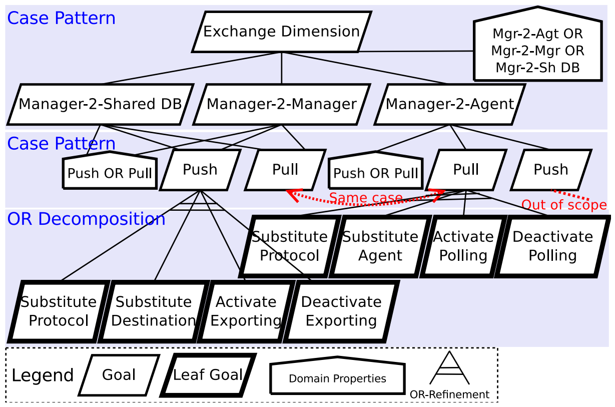

Figure 2: Exchange Dimension Pattern

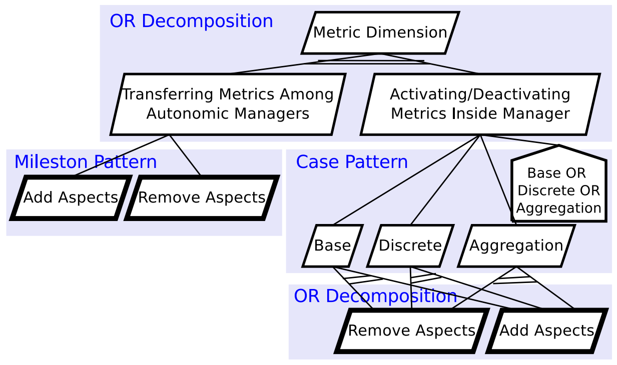

Figure 3: Metric Dimension Pattern

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). Besides identifying communication classes, we need to deal with *pull & push* communication modes. In *pull*, the entity needing information solicits the one possessing it, which responds with the queried information; where in *push*, the entity possessing the information reports it to other entities. By taking into consideration push and pull modes, along with previous communications classes, we use *case pattern* for the first two refinement levels to cover all possible cases (see Figure 2).

Based on the triplet ⟨ Information Source, Communication Protocol, Information Destination ⟩, 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.

Notice that in both Manager-2-(Manager/Shared DB) pull mode communications, the manager responding to requests is considered as agent (because it is the information source); 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.

B. Metric Dimension Pattern

Context. The main idea behind building autonomic systems is to delegate decisions, that human administrators are used to make, to autonomic systems themselves. Thus, to be able to make ”wise” decisions, the 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 more 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 (see Figure 3).

Regarding metrics manipulation inside an autonomic manager, the second refinement level uses *case pattern* to cover metric classes. Our research exploits both CIM Metric Model classifying metrics into *Base, Discrete & Aggregation*, as well as our mathematical extension [6] 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 (*Add Aspects* in Figure 3, as Subgoal 1 in Table I) first, and then removed from the delegating one (*Remove Aspects*, as Subgoal 2).

It is worth noting 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), where the latter attributes describe the instrumented values and their relevant contexts. For further information, the reader is referred to the DMTF Base Metric Profile [11].

Examples. This pattern can be applied in the following cases: (1) Performing troubleshooting, or applying root cause analysis algorithms, because they require the instrumentation of additional metrics. (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.

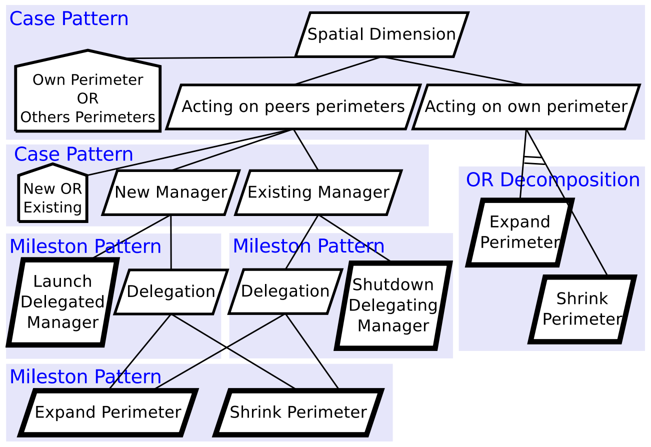

Figure 4: Spatial Dimension Pattern

C. Spatial Dimension Pattern

Context. As mentioned earlier, in an 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 the runtime. Patterns belonging to this dimension are useful to react in regard with important changes concerning the scope of managed resources.

Pattern Refinement. 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 (see Figure 4).

In fact, acting on its own perimeter is OR-decomposed using *Expand* and *Shrink Monitoring Perimeter* leaf goals. 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 4, as Subgoal 1 in Table I), and then, delegating perimeter (*Delegation*, as Subgoal 2). In turn, the delegation goal is also refined though *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).

In the second case, where acting is held on an existing manager, the refinement is done twice, first 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) Minimizing the overall monitoring charge in terms of dedicated monitoring entities.

D. Temporal Dimension Pattern

Context. Temporal aspects are decisive factors in adapting monitoring behavior. Notice that previous patterns are explained without time considerations, but in fact, they imply some temporal aspects. Patterns belonging to this dimension are useful either to overcome both temporal violations and scheduling problems, or to tune the analysis over the instrumented metrics.

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 the 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 (see Figure 5). Note that Manager-2-Agent push mode and Manager-2-(Manager/Shared DB) pull mode are not mentioned for the reasons explained in Section III-A.

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 six leaf goals OR-decomposing periodic poll (export), namely: *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 given/adjustable offset.

Regarding metrics calculation, we identify the case of modifying the temporal interval covered by the metric value. However, the validity of a metric value that is not instantaneous (*e.g.,* throughput) is equal to the temporal interval through which that value was measured. Therefore, *case pattern* is used twice to cover all possible metric classes previously mentioned. We refine only the measurable, mathematical & aggregation metrics, because time has a sense in their calculation, but not the other metrics. Thus, at the fourth refinement level, we OR-decompose measurable & mathematical metrics using *Update Time Scope Interval*. Rather, *Update Time Series Interval* OR-decomposes aggregation metrics.

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.

Notice that all previous patterns are subject to be updated and enriched, in order to integrate new monitoring adaptation actions. For instance, we can address temporal aspects of *alarms filtering* by delaying delivery of redundant alarms [12], as well as *alarms correlation* by modifying the stream interval time during which *Complex Event Processing* engines (*e.g.,* Esper & Drools) perform correlations. OR-decomposition into *Update Waiting Time* & *Update Window Time* could be used for these two cases respectively.

IV. Case-Study

Context. Our scenario takes place in a cloud data center hosting a large number of virtual machines (VMs), and pro-

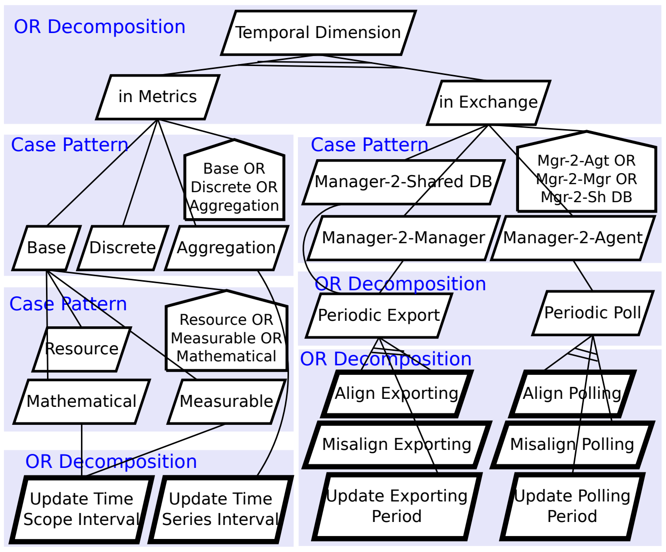

Figure 5: Temporal Dimension Pattern

viding its clients with a continuous monitoring of the enforced SLAs metrics. Each VM integrates an agent providing predetermined metrics reflecting VM healthiness. In most large scale systems, distributed agents periodically push their metrics; in our case, agents push those metrics every 10 seconds to specific pre-configured autonomic managers. To facilitate the case-study, we assume that our studied SLA template encapsulates the same metrics pushed by agents. Besides, this SLA template distinguishes two time-slots: metrics are to 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. Once a new SLA is enforced, the autonomic managers use pull mode to collect VMs metrics with the lowest freshness value (3 seconds).

Objectives. Human administrators identify two high level goals to be satisfied during the monitoring system runtime: *Respect Metrics Freshness* makes sure that SLAs are monitored appropriately, and *Minimize Monitoring Cost* aims at limiting the resources dedicated to monitoring as much as possible.

Patterns. We can exploit several patterns to deal with 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 6) with respect to the highest freshness range (6 seconds). If delivering freshness violates the highest freshness, that would be a result of overloading manager [2], 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 (*e.g.,* 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*). This adaptation necessitates applying another one belonging to *exchange pattern* to stop the pollings that are launched during the first time-slot (*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.

Autonomic managers would be able to adapt their monitoring, if they recognize adaptation stimuli. Therefore, we exploit *guard pattern* to apply adaptation actions (*Adaptation* in Figure 6, as Subgoal 2 in Table I) as response to specific stimulus (*Guard*, as Subgoal 1), while maintaining the current monitoring behavior unless adaptation takes place (*Unless*, as Subgoal 3).

V. Related Work

In this section, we try to align our approach of adapting monitoring using goal-oriented dimensional patterns with other existing trends focusing on monitoring of QoS in autonomic systems [13][14][15][16][17][18].

In order to manage QoS in autonomic systems, the latter applies adaptation actions. In many cases, for instance [13], this adaptation doesn't concern the monitoring system itself, but precisely, is applied on the managed system services and infrastructure (*i.e.,* reconfiguring resource allocation). Certainly, this adaptation will result in increased quality, but this way, the knowledge of the management system won't exceed a "maximum ceiling" and management will be limited in terms of treating new situations.

Monitoring more metrics or managed resources is addressed in [14][16][18] either to deal with the managed scope changes, or to operate a "minimal" monitoring that is able to be extended in case of SLA violations, or even to adapt monitoring to meet SLA modifications. Indeed, it is important to scale up/down monitored metrics and resources. But it isn't clear whether this capability could be applied in other scenarios for other objectives, if so, how that could be feasible.

Runtime deployment of monitoring resources (*i.e.,* managers, probes) is discussed in [14][15][17] either to integrate monitoring into the SLA management life-cycle of large scale systems, or to replace failed managers, or even to monitor some 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 (*i.e.,* planning & executing) of the distributed monitoring among several collaborating managers.

Inspired from the autonomic computing reference architecture proposed in [10], patterns regarding the distribution of the MAPE loop modules were proposed in [19][20]. Those patterns are useful in terms of design reuse as well as clarifying

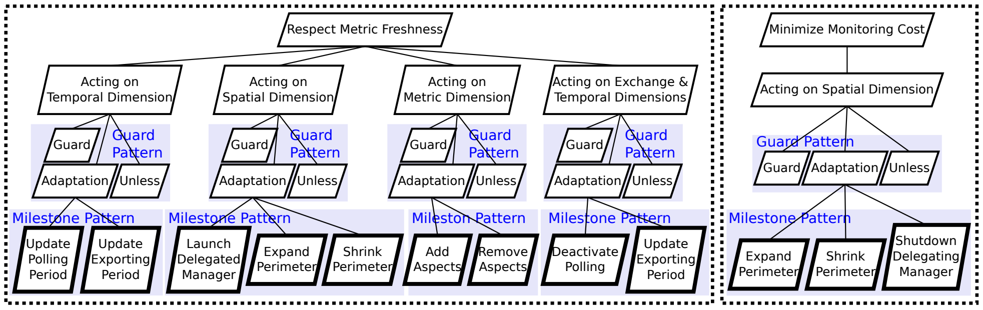

Figure 6: *Respect Metrics Freshness & Minimize Monitoring Cost* Refinement

the application contexts and benefits, but they target mainly the deployment of the monitoring modules rather than the monitoring behavior itself. In addition, they don't treat the monitoring adaptation in regard with quality requirements.

VI. Conclusion & Perspectives

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. It means that monitoring is configured starting from quality specification (*e.g.,* SLA), and reconfigured based on adaptation patterns, that are exploited to achieve high level quality objectives. We designed four monitoring adaptation patterns according to dimensions that represent various aspects on which adaptation actions can apply to. Thus, each dimension represents a "starting point" reflection to elicit monitoring goals that are refined till reaching leaf goals.

About perspectives, we are currently investigating how monitoring adaptations could influence the stability at the autonomic system whole scale, in case of applying many overlapped adaptation leaf goals over several autonomic managers. In addition, the agent side adaptations need to be investigated, and orchestrated with those applied at the autonomic manager side.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[2] A. Toueir, J. Broisin, and M. Sibilla, "A goal-oriented approach for adaptive sla monitoring: a cloud provider case study," in LATINCLOUD 2013, Maceió, Brazil, December 2013.

[3] A. Moui, T. Desprats, E. Lavinal, and M. Sibilla, "A cim-based framework to manage monitoring adaptability," in Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), 2012, pp. 261–265.

[4] A. Moui, T. Desprats, E. Lavinal, and M. Sibilla, "Information models for managing monitoring adaptation enforcement," in International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE), Nice, 22/07/2012-27/07/2012, 2012, pp. 44–50.

[5] A. Moui, T. Desprats, E. Lavinal, and M. Sibilla, "Managing polling adaptability in a cim/wbem infrastructure," in 2010 4th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM), 2010, pp. 1–6.

[6] A. Toueir, J. Broisin, and M. Sibilla, "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), 2011, pp. 1–6.

[7] A. Van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[8] A. Van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in Proceedings of the 22Nd International Conference on Software Engineering, ser. ICSE '00, 2000, pp. 5–19.

[9] R. Darimont and A. Van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," in ACM SIGSOFT Software Engineering Notes, vol. 21, no. 6. ACM, 1996, pp. 179–190.

[10] IBM Corp., "An architectural blueprint for autonomic computing," IBM White Paper, June 2005.

[11] A. Merkin, "Base metrics profile," December 2009, document Number: DSP1053.

[12] A. Clemm, Network Management Fundamentals. Cisco Press, 2006, ch. 5, pp. 138–141.

[13] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A self-adaptive hierarchical monitoring mechanism for clouds," Journal of Systems and Software, vol. 85, no. 5, 2012, pp. 1029–1041.

[14] D. Roxburgh, D. Spaven, and C. Gallen, "Monitoring as an sla-oriented consumable service for saas assurance: A prototype," in 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2011, pp. 925–939.

[15] P. Thongtra and F. Aagesen, "An adaptable capability monitoring system," in 2010 Sixth International Conference on Networking and Services (ICNS), 2010, pp. 73–80.

[16] M. Munawar, T. Reidemeister, M. Jiang, A. George, and P. Ward, "Adaptive monitoring with dynamic differential tracing-based diagnosis," in Managing Large-Scale Service Deployment, ser. Lecture Notes in Computer Science, F. Turck, W. Kellerer, and G. Kormentzas, Eds. Springer Berlin Heidelberg, 2008, vol. 5273, pp. 162–175.

[17] J. Nobre, L. Granville, A. Clemm, and A. Prieto, "Decentralized detection of sla violations using p2p technology," in Proceedings of the 8th International Conference on Network and Service Management, 2012, pp. 100–107.

[18] P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner, "Crossflow: Cross-organizational workflow management in dynamic virtual enterprises," International Journal of Computer Systems Science & Engineering, vol. 15, 2000, pp. 277–290.

[19] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. Göschka, "On patterns for decentralized control in self-adaptive systems," in Software Engineering for Self-Adaptive Systems II, R. Lemos, H. Giese, H. Müller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 76–107.

[20] A. J. Ramirez and B. H. C. Cheng, "Design patterns for developing dynamically adaptive systems," in Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, ser. SEAMS '10, 2010, pp. 49–58.