Multi-layered Monitoring and Adaptation*

Sam Guinea[1], Gabor Kecskemeti[2], Annapaola Marconi[3], and Branimir Wetzstein[4]

[1] Politecnico di Milano
Deep-SE Group - Dipartimento di Elettronica e Informazione
Piazza L. da Vinci, 32 - 20133 Milano, Italy
guinea@elet.polimi.it
[2] MTA-SZTAKI
Laboratory of Parallel and Distributed Systems
Kende u. 13-17, 1111 Budapest, Hungary
kecskemeti@sztaki.hu
[3] Fondazione Bruno Kessler
via Sommarive 18, 38123 Trento, Italy
marconi@fbk.eu
[4] University of Stuttgart
Institute of Architecture of Application Systems
Universitaetsstr. 38, 70569 Stuttgart, Germany
wetzstein@iaas.uni-stuttgart.de

Abstract. Service-based applications have become more and more multi-layered in nature, as we tend to build software as a service on top of infrastructure as a service. Most existing SOA monitoring and adaptation techniques address layer-specific issues. These techniques, if used in isolation, cannot deal with real-world domains, where changes in one layer often affect other layers, and information from multiple layers is essential in truly understanding problems and in developing comprehensive solutions.
In this paper we propose a framework that integrates layer specific monitoring and adaptation techniques, and enables multi-layered control loops in service-based systems. The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

1 Introduction

Service-based systems are built under an open-world assumption. Their functionality and quality of service depend on the services they interact with, yet these services can evolve in many ways, for better or for worse. To be sure these evolutions do not lead to systems that behave inadequately or fail, service-based

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (Network of Excellence S-Cube – http://www.s-cube-network.eu/)

systems must be able to re-arrange themselves to cope with change. A typical way of dealing with these issues is to introduce some variant of the well-known monitor-analyze-plan-execute (MAPE) loop into the system [6], effectively making the system self-adaptive.

The service abstraction has become so pervasive that we are now building systems that are multi-layered in nature. Cloud-computing allows us to build software as a service on top of a dynamic infrastructure that is also provided as a service (IaaS). This complicates the development of self-adaptive systems because the layers are intrinsically dependent one of the other. Most existing SOA monitoring and adaptation techniques address one specific functional layer at a time. This makes them inadequate in real-world domains, where changes in one layer will often affect others. If we do not consider the system as a whole we can run into different kinds of misjudgments. For example, if we witness an unexpected behavior at the software layer we may be inclined to adapt at that same layer, even though a more cost-effective solution might be found either at the infrastructure layer, or by combining adaptations at both layers. Even worse, a purely software adaptation might turn out to be useless due to infrastructural constraints we fail to consider. Similar considerations are made in case of the unexpected behavior at the infrastructure layer, or at both.

In this paper we propose a framework that integrates software and infrastructure specific monitoring and adaptation techniques, enabling multi-layered control loops in service-based systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a coordinated fashion. In our prototype we have focused on the monitoring and adaptation of BPEL processes that are deployed onto a dynamic infrastructure.

Building upon our past experiences we have integrated process and infrastructure level monitoring [2,8] with a correlation technique that makes use of complex event processing [1]. The correlated data, combined with machine-learning techniques, allow us to pinpoint where the problems lie in the multi-layered system, and where it would be more convenient to adapt [7,12]. We then build a complex adaptation strategy that may involve the software and/or the infrastructure layer [13], and enact it through appropriate effectors.

The proposed approach is evaluated on a medical imaging procedure for Computed Tomography (CT) Scans, an e-Health scenario characterized by strong dependencies between the software layer and infrastructural resources.

The rest of this paper is organized as follows. Section 2 gives a high-level overview of the integrated monitoring and adaptation framework used to enable the multi-layered control loops. Section 3 details the software and infrastructure monitoring tools and how they are correlated using complex event processing. Section 4 explains how decision trees are used to identify which parts in the system are responsible for the anomalous behaviors and what adaptations are needed. Section 5 explains how we coordinate single-layer adaptation capabilities to define a multi-layered adaptation strategy, while Section 6 presents the tools used to actually enact the adaptations. Section 7 evaluates the integrated

approach on a medical imaging procedure. Section 8 presents related work, and Section 9 concludes the paper.

2 The Integrated Monitoring and Adaptation Framework

We propose an integrated framework that allows for the installation of multi-layered control loops in service-based systems. We will start with a conceptual overview, and then provide more details on the single techniques we have integrated in our prototype.

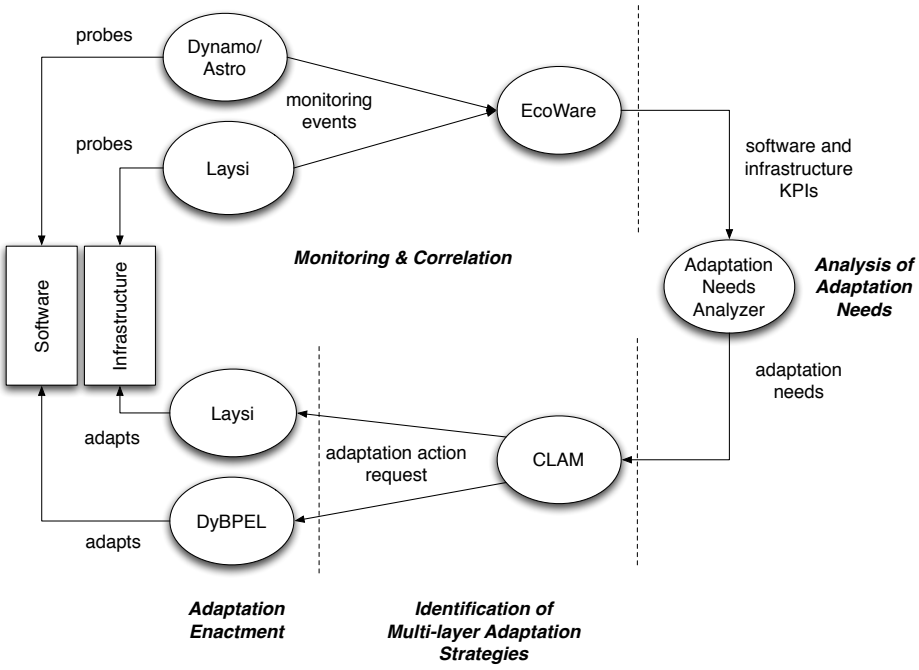

Fig. 1. The Monitoring and Adaptation Framework.

Figure 1 gives a high-level view of our integrated monitoring and adaptation framework, as used in a multi-layered software and infrastructure system. To establish self-adaptation, the framework applies a slight variation of the well-known MAPE control loop. Dashed vertical lines separate the four main steps in the loop, while oval shapes represent the concrete techniques that we have integrated – detailed later in Sections 3–6.

In the *Monitoring and Correlation* step, sensors deployed throughout the system capture run-time data about its software and infrastructural elements. The collected data are then aggregated and manipulated to produce higher-level correlated data under the form of general and domain-specific metrics. The main

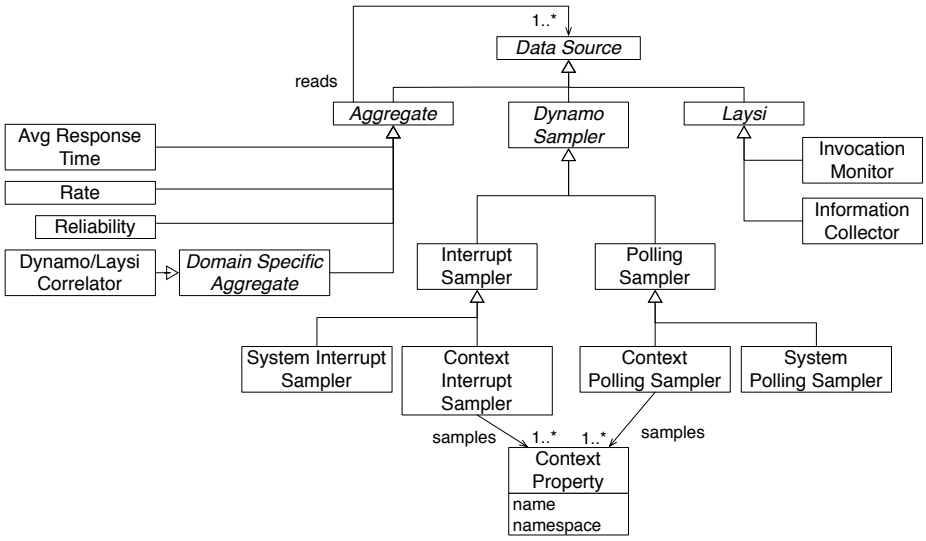

Fig. 2. The Monitoring and Correlation Model.

goal is to reveal correlations between what is being observed at the software and at the infrastructure layer to enable global system reasoning.

In the *Analysis of Adaptation Needs* step, the framework uses the correlated data to identify anomalous situations, and to pinpoint and formalize where it needs to adapt. It may be sufficient to adapt at the software or at the infrastructure layer, or we may have to adapt at both.

In the *Identification of Multi-layer Adaptation Strategies* step, the framework is aware of the adaptation capabilities that exist within the system. It uses this knowledge to define a multi-layer adaptation strategy as a set of software and/or infrastructure adaptation actions to enact. A strategy determines both the order of these actions and the data they need to exchange to accomplish their goals.

In the *Adaptation Enactment* step, different adaptation engines, both at the software and the infrastructure layer, enact their corresponding parts of the multi-layer strategy. Each engine typically contains a number of specific modules targeting different atomic adaptation capabilities.

3 Monitoring and Correlation

Monitoring consists in collecting data from a running application so that they can be analyzed to discover runtime anomalies; event correlation is used to aggregate runtime data coming from different sources to produce information at a higher level of abstraction. In our integrated framework we can obtain low-level data/events from the process or from the context of execution using Dynamo [2], or from the infrastructure using Laysi [8]. We can then manipulate the data to

obtain higher-level information using the event correlation capabilities provided by EcoWare [1]. Figure 2 gives an overview of the kind of data sources available through Dynamo, Laysi, and EcoWare.

Dynamo provides means for gathering events regarding either (*i*) a process' internal state, or (*ii*) context data[5]. `Interrupt Sampler`s interrupt a process at a specific point in its execution to gather the information, while `Polling Sampler`s do not block the process but gather their data through polling.

The `Invocation Monitor` is responsible for producing low-level infrastructure events through the observation of the various IaaS systems managed by Laysi. These events signal a service invocation's failure or success, where failures are due to infrastructure errors. The infrastructure, however, can also be queried through the `Information Collector` to better understand how services are assigned to hosts. The differences between the utilized infrastructures and the represented information are hidden by the information collector component of the MetaBroker service in Laysi.

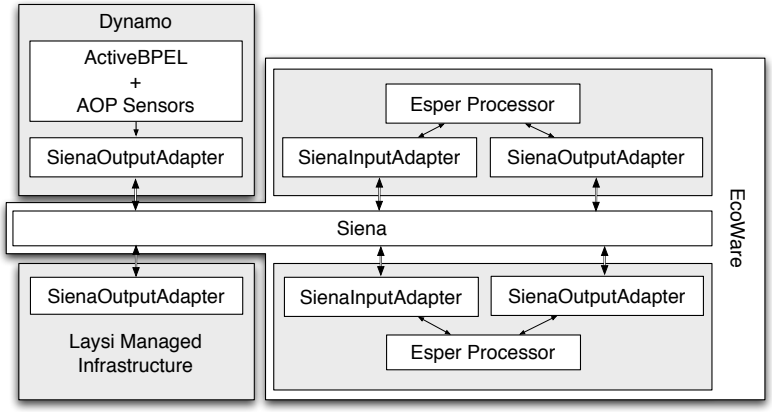

Fig. 3. The Dynamo and EcoWare Architecture.

The events collected through Dynamo and Laysi can be further aggregated or manipulated by EcoWare. We can use a predefined aggregate metric such as `Reliability`, `Average Response Time`, or `Rate`, or we can use a domain-specific aggregate whose semantics is expressed using the Esper event processing language. Aggregates process events coming from one or more data sources and produce new ones that can be even further manipulated in a pipe-and-filter style.

For our integrated approach we developed a domain-specific aggregate called the `Dynamo/Laysi Correlator` to correlate events produced at the software and the infrastructure layers. This component exploits a correlation data set that is

[5] We intend as context any data source, external to the system, that offers a service interface.

artificially introduced by Dynamo in every service call it makes to the Laysi infrastructure. The correlation data contains the name of the process making the call to Laysi, the invocation descriptor in the form of a unique JSDL (Job Submission Description Language) document, and a unique ID for the process instance that is actually making the request. These data are also placed within the events that are generated by the Invocation Monitor, allowing EcoWare to easily understand which software- and infrastructure-level events are related. Figure 3 gives an overview of the technical integration of Dynamo, Laysi, and EcoWare, which is achieved using a Siena publish and subscribe event bus. Input and output adapters are used to align Dynamo, Laysi, and the event processors with a normalized message format.

4 Analysis of Adaptation Needs

Monitoring and correlation produce simple and complex metrics that need to be evaluated. A Key Performance Indicator consists of one of these metrics (e.g., overall process duration) and a target value function which maps values of that metric to two or more categories on a nominal scale (e.g., "process duration < 3 days is *good*, otherwise *bad*" defines two KPI categories). These KPI categories allow us to interpret whether, and how, KPI metric values conform to business goals. If monitoring shows that many process instances have bad KPI performance, we need to (*i*) analyze the influential factors that lead to these bad KPI values, and (*ii*) find adaptation actions that can improve those factors and thus the KPI. Figure 4 shows an overview of the KPI-based Adaptation Needs Analyzer Framework [7,12] and its relation to the overall approach. It consists of two main components: an `Influential Factor Analysis` component and an `Adaptation Needs Analysis` component.

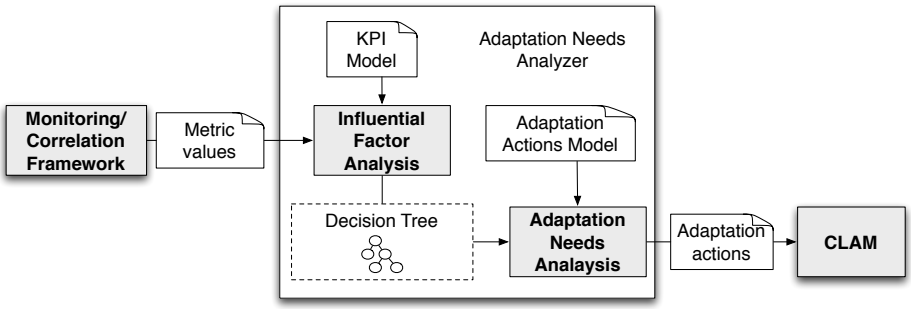

Fig. 4. Adaptation Needs Analysis Framework.

The Influential Factor Analysis receives the metric values for a set of process instances within a certain time period. In this context, interesting metrics are measured both on the process level and the service infrastructure level. At the

process level, metrics include the durations of external service calls, the duration of the overall business process, the process paths taken, the number of iterations in loops, and the process' data values. Service infrastructure metrics describe the service invocation properties which include the status of the service invocation (successful, failed), and properties such as the infrastructure node on which the service execution has been performed.

It uses machine learning techniques (decision trees) to find out the relations between a set of metrics (potential influential factors) and the KPI category based on historical process instances [12]. The algorithm is fed with a data set, whereby each data item in this set represents one process instance and the values of all the metrics that were measured for that instance and the KPI category that has been evaluated. The algorithm creates a decision tree in which nodes represent metrics (e.g., the duration of a particular activity), outgoing edges represent conditions on the values of the metric, and leaves represent KPI categories. By following the paths from the root of the tree to its leaves, we can see for which combinations of metrics and values particular KPI categories have been reached (e.g., if duration of activity A was above 3 hours and activity B was executed on node 2 the KPI value was bad).

Based on this analysis the next step is to use the Adaptation Needs Analysis component to identify the adaptation needs, i.e., what is to be adapted in order to improve the KPI [7]. The inputs to this step are the decision tree and an adaptation actions model which has to be manually created by the user. The model contains different adaptation actions, whereby each specifies an adaptation mechanism (e.g., service substitution, process structure change) and how it affects one or more of the metrics used in the Influential Factor Analysis. For example, an adaptation action could be to substitute service A in the process with service B, and its effect could be "service response time < 2 h". The Adaptation Needs Analysis extracts the paths which lead to bad KPI categories from the tree and combines them with available adaptation actions which can improve the corresponding metrics on the path. As a result, we obtain different sets of potential adaptation actions. However, each of these sets does not yet take cross-layer dependencies between adaptation actions into account. This is performed in the next step by the CLAM framework.

5 Identification of Multi-layer Adaptation Strategies

The main aim of the Cross Layer Adaptation Manager (CLAM) [13] is to manage the impact of adaptation actions across the system's multiple layers. This is achieved in two ways: on the one hand CLAM identifies the application components that are affected by the adaptation actions, and on the other hand, it identifies an adaptation strategy that properly coordinates the layer-specific adaptation capabilities. CLAM relies on a model of the multi-layer application that contains the current configuration of the application's components (e.g. business processes with KPIs, available services with stated QoS and general information, available infrastructure resources) and their dependencies (e.g. business activity

A is performed by service S). When the CLAM identifies the components that are affected by the adaptation actions, it uses a set of `checkers`, each associated with a specific application concern (e.g. service composition, service performances, infrastructure resources), to analyze whether the updated application model is compatible with the concern's requirements. The goal is to produce a strategy that is modeled as an Adaptation Tree. The tree's root represents the model's initial configuration; its other nodes contain the configurations of the model, as updated by the adaptation actions, and the checkers that need to be invoked at each step; its edges represent the outcome of the invoked checkers.

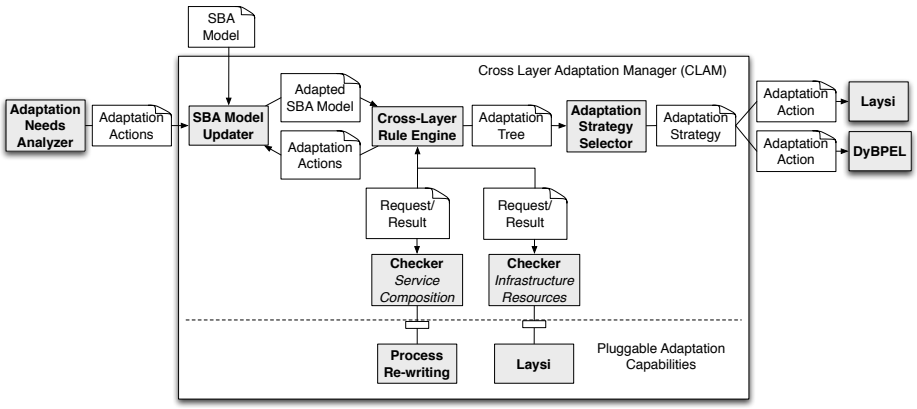

Fig. 5. CLAM: Cross-layer Adaptation Manager

Figure 5 presents an overview of CLAM's architecture. Whenever a new set of adaptation actions is received from the Adaptation Needs Analyzer, the `SBA Model Updater` module updates the current application model by applying the received adaptation actions. CLAM requires that all the adaptation actions be applicable with respect to the current model. However, this is guaranteed in the proposed multi-layer framework by the Adaptation Needs Analyzer.

The adapted model is then used by the `Cross-layer Rule Engine` to detect the components in the layers affected by the adaptation and to identify, through the set of predefined rules, the associated adaptation checkers. If some constraints are violated, the checker is responsible for searching for a local solution to the problem. This analysis may result in a new adaptation action to be triggered. This is determined through the interaction with a set of pluggable application-specific adaptation capabilities.

The Cross-layer Rule Engine uses each checker's outcome to progressively update the strategy tree. If the checker triggers a new adaptation action, the Cross-layer Rule Engine obtains a new adapted model from the Model Updater, and adds it as a new node to the strategy tree, together with the new checkers to be invoked. If the checker reports that the adaptation is not compatible and

that no solution can be found, the node is marked as a red leaf; the path in the tree that leads from the root to that specific node represents an unsuccessful strategy. On the contrary, if all checks complete successfully, the node is a green leaf that can be considered a stable configuration of the application, and the corresponding path in the tree represents an adaptation strategy that can be enacted.

If multiple adaptation strategies are identified, the `Adaptation Strategy Selector` is responsible of choosing the best strategy by evaluating and ranking the different strategies according to a set of predefined metrics. The selected strategy is then enacted passing the adaptation actions to the adaptation enactment tools.

Due to our scenario's requirements we have currently integrated two specific adaptation capabilities into our framework: the `Process Re-Writing` planner, responsible of optimizing service compositions by properly parallelizing sequential activities, and `Laysi`, whose aim is to guarantee a correct and optimized usage of infrastructure resources.

The Process Re-writing Planner is an adaptation mechanism that, given a BPEL process and a set of optimization requirements, automatically computes a new version of the process that maximizes the parallel execution of activities. This is done taking into account a set of data and control flow requirements that characterize the process' correct behavior (e.g. activity A cannot be executed in parallel with B, activity A must follow activity B, etc.), as well as any interaction protocols the partner services may require (e.g. if service S expects activity A to be executed before activity B, than this protocol requirement will be satisfied).

Laysi offers self-management capabilities for service infrastructures and allows new infrastructure level requirements to be evaluated before the actual service invocations take place. Hence, upon receiving the possible parallel execution options from Process Re-writing Planner, the CLAM architecture presents these options as requirements (including the required parallelism and time constraints) to Laysi for all the not-yet executed service calls. In response, Laysi determines the feasibility of the proposed requirements taking into account that a rearrangement of the service infrastructure may be needed. If the system decides to enact the adaptation and the infrastructure needs to be rearranged, Laysi will ensure the next invocation can meet its agreed constraints according to the adaptation enactment tasks specified in the following section.

6 Adaptation Enactment

In our integrated approach we enact software adaptations through `DyBPEL`, and infrastructure adaptations through `Laysi`. CLAM issues specific actions of the chosen adaptation strategy to each tool in a coordinated fashion.

In the proposed integration, DyBPEL is responsible of enacting the process restructuring adaptations identified by the Process Re-writing Planner.

DyBPEL extends an open-source BPEL execution engine (ActiveBPEL) with the capability to modify a process' structure at run time. The change can be

applied to a single process instance or to an entire class of processes. DyBPEL consists of two main components: a `Process Runtime Modifier`, and a `Static BPEL Modifier`. The runtime modifier makes use of AOP techniques to intercept a running process and modify it in one of three ways: by intervening on its BPEL activities, on its set of partnerlinks, or on its internal state. The runtime modifier takes three parameters. The first is an XPath expression that uniquely identifies the point in the process execution in which the restructuring has to be activated. The second is an XPath expression that uniquely identifies the point in the process in which restructuring needs to be achieved (it can be different than the point in which the restructuring is activated). The third is a list of restructuring actions. Supported actions consist of the addition, removal, or modification of BPEL activities, partnerlinks, and data values. When dealing with BPEL activities we must provide the BPEL snippet that needs to be added to the process, or used to modify one of the process' existing activities. When dealing with partnerlinks we must provide the new partnerlink that needs to be added to the process, or used to modify an existing one. When dealing with the process' state we must uniquely identify a BPEL variable within the process to be added or modified, and the XML snippet that will consist of its new value.

When the process restructuring needs to be more extensive, we can use the static BPEL modifier. It supports the same kinds of modifications to the process' activities, partnerlinks, and internal variables, except that the modifications are performed on the process' XML definition. This operation is completely transparent to users. First of all, already running instances are not modified and changes are only applied to new instances. Second, using the same endpoint, all new process requests are forwarded to the newly deployed version of the process.

Regarding infrastructure adaptation, Laysi always performs service requests on a best-effort basis. Each service invocation is handled individually and the various calls are assumed to be independent. Consequently, the performance of the service requests might not be aligned with the higher layers of the service-based system. To provide better alignment with the service composition layer we can specify special constraints about service placement (e.g. service instance A should be hosted within the same provider as service instance B) and availability within the infrastructure (e.g. a service instance should be available before the invocation request is placed in the call queue of Laysi). These constraints are derived directly from the business process and the future interactions between the available service instances hosted by the infrastructure. Laysi constructs the service infrastructure on five layers: meta negotiators, meta brokers, service brokers, automatic service deployers, and the physical infrastructures (grid resources or cloud based virtual machines). These infrastructure layers autonomously adapt themselves to the placed constraints (e.g. placement, availability, CPU, memory, pricing). The autonomous behavior of the infrastructure may involve (*i*) new service instance deployment in high demand situations, (*ii*) service broker replacement in case of broken or low performing physical infrastructures, and/or (*iii*) negotiation bootstrapping if a new negotiation technique is required.

7 The CT Scan Scenario

The application domain considered in this paper concerns the medical imaging procedure for Computed Tomography (CT) Scans. A CT Scan is an X-ray based medical test that, exploiting sophisticated image processing algorithms, produces cross-sectional images of the inside of the body. These images can be further processed to obtain three dimensional views.

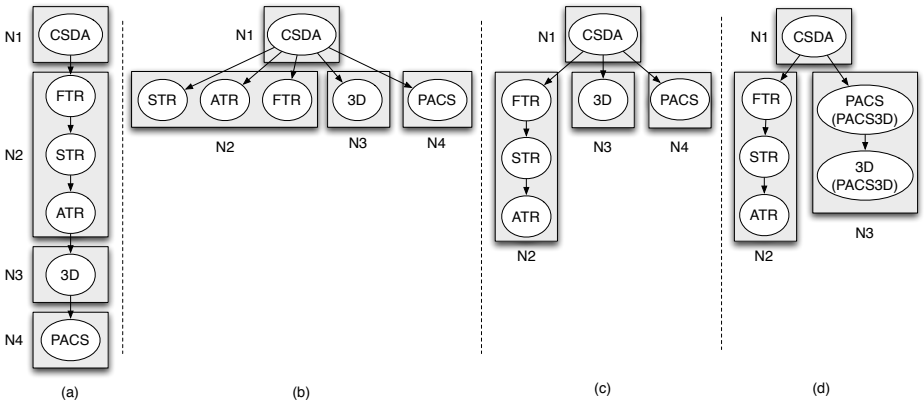

Fig. 6. Evolution of the CT scan scenario.

Figure 6(a) describes the typical CT Scan process. White ovals represent software services, while gray rectangles tell us the infrastructure nodes hosting them. During the Cross Sectional Data Acquisition phase (service `CSDA`) the CT scanner acquires X-ray data for individual body cross sections depending on which parts of the body need to be scanned. These data are then used by complex image processing services (offered by various hosts in the infrastructure) to obtain a set of cross-sectional images from different perspectives as well as 3D volumetric information. The services are the Frontal Tomographic Reconstruction service (`FTR`), the Sagittal Tomographic Reconstruction service (`STR`), the Axial Tomographic Reconstruction service (`ATR`), and the 3D volumetric information service (`3D`). Finally, the data is stored to a picture archiving and communication system using the `PACS` service.

These activities require enormous processing power. To keep costs down, the hospital only maintains the resources needed for emergency CT scans. During burst periods, such as during the public opening hours of the CT laboratory, it relies on an infrastructure dynamically extensible with virtual machines from IaaS cloud infrastructures managed by Laysi.

In the following we show how our approach can be used to automatically adapt this multi-layered system. The CT Scan process is initially designed by a domain expert on the basis of the common medical procedure. The obtained process is a simple sequence of actions that does not embed any optimization with

respect to its performance (Figure 6 (a)). The domain expert also specifies his goals for the quality of the medical procedure using a set of KPIs. For instance, an important goal is to ensure that the processing time of a CT scan does not rise above 60 minutes. An advanced user, such as a hospital IT technician, defines a set of adaptation actions that can be used to improve the process' performance: (*i*) the parallelization of process activities; (*ii*) the substitution of some services (for example, the use of a more costly PACS3D service capable of substituting both services PACS and 3D); (*iii*) the deployment of a service onto a new infrastructural node with specific characteristics.

At run time we collect monitoring events both at the software and the infrastructure level and correlate them using EcoWare. After a certain period of time, we notice that the CT process' performance is degrading: in the last 400 scans about 25% have not achieved their desired overall CT scan processing time. In order to identify the reasons for this behavior, the Influential Factor Analysis is fed the following process and infrastructure level metrics: (*i*) the duration of each process activity; (*ii*) the duration of the whole process with respect to the *the type of the CT scan* (whole body, head, kidney etc.) as it determines the amount of work to be done; (*iii*) the particular infrastructure node a service execution has been executed on; (*iv*) the status of a service execution (successful, faulted); and (*v*) the type of infrastructure the services have been executed on (internal or external – available through Laysi).

The Influential Factor Analysis shows that from the 100 scans which violated the KPI target, 90 scans have been "whole body CT scans" executed on an external infrastructure. It also shows that the infrastructure has caused service execution faults only in 12 cases (out of 400). Finally, all scans performed on the internal infrastructure were successful. Based on this analysis, the Adaptation Needs Analysis selects predefined adaptation actions which can improve the "overall process duration in case of whole body CT scans". It selects process activity parallelization as it is the only adaptation which has been specified to have a direct positive effect on this metric.

This adaptation action is passed to the CLAM which updates the process model so that all activities are executed in parallel. The Cross-Layer Rule Engine detects the change in the process model and understands that these changes have to be checked by the composition checker and by the infrastructure checker (as the parallel execution of services has to be supported by the underlying infrastructure). The composition checker invokes the Process Re-Writing Planner which considers the original data- and control-flows of the process. It notices that the activities cannot all be executed in parallel since five of them depend on CSDA's results; thus the planner returns a new adaptation action which ensures that CSDA is executed first, while all the other activities are conducted in parallel. The model is updated in CLAM as shown in Figure 6 (b) and a new node is added to the adaptation tree. In the next step, the Cross-Layer Rule Engine invokes the infrastructure checker component which, through Laysi, discovers that the activities for tomographic reconstruction (i.e. FTR, STR, and ATR) can only be executed on the node N2. The Rule Engine handles this

new adaptation need by invoking again the Process Re-Writing Planner with a new set of control-flow constraints (i.e. FTR, STR, and ATR must be executed sequentially). The resulting process structure is shown in Figure 6 (c), in which, after CSDA, there are three parallel branches. In one of these branches FTR, STR, and ATR are executed sequentially, while, in the other two, the process executes the 3D and PACS services. The model is updated and the infrastructure checker component is invoked again. This new version of the process passes the infrastructure validation and, since there are no more checkers to be invoked, the corresponding strategy is enacted. In particular, the adapted process is handed over to DyBPEL, which manages the transition to the new process definition.

The adapted process is executed and after a certain period of time we notice that the number of KPI violations has been reduced to 10%, and that most KPI violations happen when the PACS service's execution time is too high. Therefore, two alternative adaptation actions are found: either (*i*) move the PACS service instance to another (better performing) node, or (*ii*) replace PACS with the new service PACS3D. Both alternatives are passed to CLAM.

The CLAM Rule Manager invokes the infrastructure checker with the constraint to the Laysi infrastructure stating that the PACS service should never be executed on node N4. Unfortunately, Laysi responds that, due to constraints, this is not possible and that PACS must always be executed on N4. The CLAM Rule Manager drops the first adaptation action alternative as it is not realizable, and repeats the procedure with the second adaptation action, the substitution of the PACS service with a service called PACS3D, capable of providing both storage and 3D reconstruction at a higher cost. This alternative has to be checked both by the composition checker and by the infrastructure checker. The Process Re-writing Planner detects that a new process restructuring is necessary: a new control-flow requirement is introduced by the protocol of the PACS3D service which requires to receive and store all the X-Ray data information (PACS) before computing the 3D Scan (3D). The SBA Model resulting from this new adaptation action is depicted in Figure 6 (d). The parallel branches are now only two, one for FTR, STR, and ATR, and one for PACS3D which is called twice, once to perform 3D reconstruction, and once to perform storage. The infrastructure checker validates the new model and the corresponding strategy is enacted.

8 Related Work

There are not many approaches in literature that integrate multi-layered monitoring and adaptation of service-based systems. There are however many that focus on layer-specific problems. For example, Moser et al. [10] present VieDAME, a non-intrusive approach to the monitoring of BPEL processes. The approach accumulates runtime data to calculate QoS values such as response time, accuracy, or availability. It also provides a dynamic adaptation and message mediation service for partnerlinks, using XSLT or regular expressions to transform messages accordingly. Colombo et al. [3] extend the BPEL composition language with policy (re)binding rules written in the Drools language. These rules take the form

of if-then-else statements, allowing service bindings to depend on process data collected at run time. The approach also provides mediation capabilities through a special-purpose mediation scripting language.

Researchers that do consider multi-layered applications, on the other hand, tend to concentrate either on monitoring them or on adapting them. We present the most prominent research being done in both these fields. Foster et al. [5] have proposed an extensible framework for monitoring business, software, and infrastructure services. The framework allows different kinds of reasoners, tailored to different kinds of services, to be integrated and to collaborate to monitor decomposable service level agreement terms and expressions. The framework automatically assigns the decomposed atomic terms to specific reasoners, yet the approach does not support the correlation of terms monitored at different layers. Mos et al. [9] propose a multi-layered monitoring approach that considers service and infrastructure level events produced by services deployed to a distributed enterprise service bus. Basic computations can be performed on the events to produce aggregate information (e.g., averages) or complex event processing can be used for more complex correlations and verifications. The resulting data are analyzed by comparing them to thresholds, and the knowledge collected at the various levels are presented through appropriately differentiated user interfaces and visualization techniques. The approach does not correlate knowledge collected at the different levels.

Regarding multi-level adaptation, Efstratiou et al. [4] present an approach for adapting multiple applications that share common resources. These applications are not composed, but rather single entities affected by the same contextual attributes. Since these applications live in the same space they need to coordinate how they manage the shared resources to avoid conflicts. However, they expect the users to perceive and model the conflicts manually. Finally, Popescu et al. [11] propose a framework for multi-layer adaptation of service-based systems comprised of organization, coordination and service layers. In this approach a designer needs to prepare a taxonomy of the adaptation mismatches, and then a set of adaptation templates, known as patterns, that define generic solutions for these mismatches. This differs from our proposed approach since we do not require on design-time knowledge but discover our strategies on-the-fly.

9 Conclusion and Future Work

In this paper we have presented an integrated approach for monitoring and adapting multi-layered service-based systems. The approach is based on a variant of the well-known MAPE control loops that are typical in autonomic systems. All the steps in the control loop acknowledge the multi-faceted nature of the system, ensuring that we always reason holistically, and adapt the system in a cross-layered and coordinated fashion. We have also presented initial validation of the approach on a dynamic CT scan scenario.

In our future work we will continue to evaluate the approach through new application scenarios, and through the addition of new adaptation capabilities

and adaptation enacting techniques. We will also integrate additional kinds of layers, such as a platforms, typically seen in cloud computing setups, and business layers. This will also require the development of new specialized monitors and adaptations. Finally, we will study the feasibility of managing different kinds of KPI constraints.

References

1. L. Baresi, M. Caporuscio, C. Ghezzi, and S. Guinea. Model-Driven Management of Services. In *Proceedings of the Eighth European Conference on Web Services, ECOWS*, pages 147–154. IEEE Computer Society, 2010.
2. L. Baresi and S. Guinea. Self-Supervising BPEL Processes. *IEEE Trans. Software Engineering*, 37(2):247–263, 2011.
3. M. Colombo, E. D. Nitto, and M. Mauri. SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. In *Proceedings of the Fourth International Conference on Service Oriented Computing, ICSOC*, pages 191–202, 2006.
4. C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. An architecture for the effective support of adaptive context-aware applications. In *Proceedings of the Second International Conference on Mobile Data Management, MDM*, pages 15–26. Springer, 2001.
5. H. Foster and G. Spanoudakis. SMaRT: a Workbench for Reporting the Monitorability of Services from SLAs. In *Proceedings of the 3rd International Workshop on Principles of Engineering Service-oriented Systems, PESOS*, pages 36–42. ACM, 2011.
6. P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. *IBM TJ Watson Labs.*, October 2001.
7. R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann. Adaptation of Service-Based Applications Based on Process Quality Factor Analysis. In *ICSOC/ServiceWave Workshops*, pages 395–404, 2010.
8. A. Kertész, G. Kecskemeti, and I. Brandic. Autonomic SLA-Aware Service Virtualization for Distributed Systems. In *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP*, pages 503–510, 2011.
9. A. Mos, C. Pedrinaci, G. A. Rey, J. M. Gomez, D. Liu, G. Vaudaux-Ruth, and S. Quaireau. Multi-level Monitoring and Analysis of Web-Scale Service based Applications. In *ICSOC/ServiceWave Workshops*, pages 269–282, 2009.
10. O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In *Proceeding of the 17th international conference on World Wide Web, WWW*, pages 815–824. ACM, 2008.
11. R. Popescu, A. Staikopoulos, P. Liu, A. Brogi, and S. Clarke. Taxonomy-Driven Adaptation of Multi-layer Applications Using Templates. In *Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO*, pages 213–222, 2010.
12. B. Wetzstein, P. Leitner, F. Rosenberg, S. Dustdar, and F. Leymann. Identifying Influential Factors of Business Process Performance using Dependency Analysis. *Enterprise IS*, 5(1):79–98, 2011.
13. A. Zengin, R. Kazhamiakin, and M. Pistore. CLAM: Cross-layer Management of Adaptation Decisions for Service-Based Applications. In *Proceedings of the 9th International Conference on Web Services, ICWS*, 2011.