Using Process Technology to Control and Coordinate Software Adaptation

Giuseppe Valetto
Telecom Italia Lab and Columbia University
Giuseppe.Valetto@tilab.com

Gail Kaiser
Columbia University
kaiser@cs.columbia.edu

Abstract

We have developed an infrastructure for end-to-end run-time monitoring, behavior/performance analysis, and dynamic adaptation of distributed software. This infrastructure is primarily targeted to pre-existing systems and thus operates outside the target application, without making assumptions about the target's implementation, internal communication/computation mechanisms, source code availability, etc. This paper assumes the existence of the monitoring and analysis components, presented elsewhere, and focuses on the mechanisms used to control and coordinate possibly complex repairs/reconfigurations to the target system. These mechanisms require lower-level effectors somehow attached to the target system, so we briefly sketch one such facility (elaborated elsewhere). Our main contribution is the model, architecture, and implementation of Workflakes*, the decentralized process engine we use to tailor, control, coordinate, etc. a cohort of such effectors. We have validated the* Workflakes *approach with case studies in several application domains. Due to space restrictions we concentrate primarily on one case study, briefly discuss a second, and only sketch others.*

1. Introduction

Distributed computing is becoming (or has become) a commodity. Users rely upon distributed systems for value-added services that pervade their everyday lives, such as Web-based collaboration, electronic B2B and B2C, on-demand multimedia content, ubiquitous personal messaging, and many others. These services are often built on top of a networking infrastructure as distributed systems, often constructed by composition (that is, the servers themselves are distributed amongst components, not just the client viz-a-viz server). The complexity of the behavior and interrelationships of these "systems of systems" becomes increasingly harder to analyze in advance, and keep under control in the field. That aggravates the critical problems of managing the provisioning of the service and maintaining the intended application-level, "soft" quality of service (QoS). In order to resolve poor performance or failures, often a service is interrupted, the underlying application is taken down (at least in part), and the spiral of software lifecycle iterates back to the installation or deployment phase, and sometimes even to earlier development phases.

While such a drastic response may be obligatory at times, it is desirable when possible to resolve problems with lesser impacts and costs – while the system is running and without taking it offline. This motivates the notion of *dynamic adaptation* of complex, distributed software systems and services. By that term we mean any automated and concerted set of actions aimed at modifying, at runtime, the structure, behavior and/or performance of a target software system, typically in response to the occurrence and recognition of some (adverse) conditions. Examples range from tuning functioning parameters within a component in order to optimize its performance, to component(s) migration, to architecture-wide interventions, such as on-the-fly deployment or evolution of the service as a whole.

The dynamic adaptation theme is gaining attention as an approach to addressing the ever-increasing complexity of IT infrastructures and applications. For example, the autonomic computing initiative announced by IBM [1] and the Recovery Oriented Computing work at Berkeley and Stanford [2] strongly push in that direction. But dynamic adaptation brings about several major challenges, since it can be seen as a form of automated maintenance of "live" systems – likely to be even more complex than conventional off-line maintenance.

The hardwiring of self-adaptation provisions within the application itself is still the most common approach to dynamic adaptation, but that is feasible principally only for "new" systems, or systems whose components are under the complete control of the developers. Moreover, those hardwired provisions tend to increase the overall complexity of the system, in fact intensifying maintenance difficulties, and are often developed custom, with little reuse possible across applications or domains.

For these reasons, our research has focused on solutions orthogonal to the target system's main computation, control and communication, constituting an *externalized* dynamic adaptation infrastructure. Our

approach enables retrofitting legacy systems with the desired reconfiguration, self-healing, self-management, etc. capabilities, as well as similarly aiding systems built by third-party composition.

Our model for externalized dynamic adaptation is that of a layered architecture, comprised of layers for data collection, information analysis, decision/control and actuation (we use the terms *effectors* and *actuators* interchangeably for the units that perform this duty). This model is sketched briefly in section 2; more in-depth discussion of the externalized infrastructure as a whole, and in particular how its collection and analysis components have been fulfilled by our *Kinesthetics eXtreme (KX)* implementation, can be found in [18] [23].

This paper elaborates instead primarily on the decision and control role; in Section 3, we present how we have addressed it with process technology: We employ a process engine, called *Workflakes*, to coordinate the actuation layer. As with collection and analysis, many actuation approaches are possible; Workflakes integrates the *Worklets* mobile agents platform, which was originally developed for unrelated purposes and is described in [19]. A preliminary paper [5] sketched the basic ideas of the Workflakes project. This paper provides the first complete presentation and evaluation of the Workflakes model and architecture, according to our recently completed operational implementation.

KX, with and without Workflakes, has been employed in a number of case studies. We are concerned in this paper only with describing and evaluating the decision/control and actuation-coordination aspects of those target applications, through Workflakes, not the monitoring facilities or actuator details, addressed elsewhere. We present one such case study in full in Section 4 - on an industrial Internet service - with its most recent results, which subsume and update a previous report [26]. We also sum up some other case studies in their most interesting traits. Section 5 discusses related work specifically with respect to Workflakes, since as noted the other components of KX are presented elsewhere. Section 6 concludes the paper.

2. Externalized dynamic adaptation

An externalized dynamic adaptation platform can be seen at the highest level of abstraction as a feedback loop that is superimposed onto an existing distributed system for the purposes of continually monitoring and modifying its configuration, activity and performance. Since the feedback loop is handled outside of the target application, it is possible to maintain a clear separation between the reusable, common adaptation mechanisms and the target system specifics.

Furthermore, in order to be generally applicable in diverse usage and technological contexts, the infrastructure must be constructed with great attention to its interoperability with a variety of adaptation targets. Such generality in turn can be achieved via standardization of the interactions between the infrastructure components,. That enables to choose among the numerous technological options that can be used to implement the roles of probes, gauges, controllers and effectors, and to accommodate more easily within the model those that best suit the target system.

As a consequence, within the DARPA DASADA program [3] under which we conducted this research, a standard model - the proposed Common DASADA Infrastructure [25] - has emerged, which organizes the design of the feedback loop according to multiple, well-separated layers, as shown in Figure 1. The Collection layer first gathers information from the running target system, by instrumenting it with minimally invasive *probes* that report via a Probe Bus to the Interpretation layer. There, information is mapped and evaluated by *gauges*, and findings are reported to the Gauge Bus. Then the Decision and Control layer analyzes the implications of the gauge findings on the target system functioning and performance, and makes decisions on whether to carry out some dynamic adaptation(s). Adaptation actions would be performed by effectors at the Actuation layer, under the coordination of one or more *controllers*. Implementation-level *effectors* would thus adapt (i.e., reconfigure or tune) individual components, as well as connectors and other substructures of the target system.

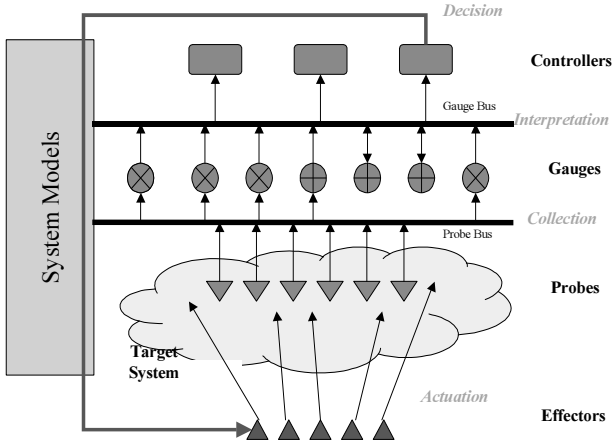

Figure 1: Dynamic adaptation infrastructure

A paramount role in this architecture is played by formal (i.e., machine-readable) models of the target system. Formal knowledge is pervasively used to drive the target instrumentation with probes, the interpretation of collected information by gauges, the decisions taken by controllers, and their coordination of the effectors'

actions. That knowledge can be captured and made available to the dynamic adaptation architecture via suitable notations and models. Those models must encompass numerous aspects of the target system, like functional and non-functional properties, protocols, architecture, distribution, etc. However, dynamic adaptation does not require complete *a priori* target analysis and modeling: Models can instead be developed piecemeal and selectively, with respect to those target substructures and facets of relevance to each dynamic adaptation application.

At this level of abstraction, we are not concerned so much with indicating what form of modeling is the most suitable for any or all of those purposes: it is indeed possible that various concerns are better captured not by a single, but by multiple complementary representations. Furthermore, even proprietary or internalized models can serve well the dynamic adaptation facilities: developing exchangeable models in some "standard" formalism – as well as leveraging any pre-existing codified knowledge - is not necessary, although certainly desirable. Whereas further discussion of the modeling issues connected to dynamic adaptation is outside the scope of this paper, some indications originating from our experience are reported in the remainder, in particular related to the use of Architecture Description Language (ADL) tools.

Notice that while the architecture proposed above is largely decoupled from the running target system, this is not to say that the specific probes, gauges, controllers, effectors and the models that govern them are themselves independent of the target system. In fact, probes and effectors must often be specialized to the implementation technology; gauges and decision mechanisms must be specialized to the problem logic and the environment.

Notice also how the layered design facilitates the clear separation of the various kinds of functionality taking part in the overall infrastructure (a similar separation of concerns is advocated in other dynamic adaptation initiatives, such as [30]). In the DARPA DASADA program, much work by ourselves and others has been devoted to the development of proposals for standard probing [4] and gauging [20] APIs. The decision and control roles in a dynamic adaptation platform are however less well understood, thus further from standardization.

Finally, to effectively close the loop represented in Figure 1, the infrastructure must automate decisions on the adaptation to be carried out, orchestrate each adaptation as it occurs, and provide adequate effectors to actuate that adaptation via appropriate side effects on the target system. Workflakes explores part of that problem space, in particular how to express control, and how to organize and exert it on multiple coordinated effectors.

3. The Workflakes approach

The output of gauges represents the input to a decision process that determines whether/how the target system must be adapted. In the simplest case, some gauge may assert a fact that already carries with it unequivocally defined consequences. Other times, a variety of tools could be exploited for decision support: for example, formal architectural knowledge models of the target system, coupled with constraint analysis and architecture transformation tools, such as [16] [22], in our experience can be very effective in driving the decision and actuation phases of dynamic adaptation.

When a decision to apply some adaptation is made, a single action will sometimes suffice to fulfill it (this is the assumption of systems such as [35]). In most cases, however, the adaptation will have to be mapped onto several finer-grained and concerted activities, impacting several implementation-level elements. Then a sophisticated coordination mechanism is needed: some of those activities may be conditional, or dependent on others, or may fail, calling for contingency planning, etc.

To address that complexity, Workflakes relies on process-based coordination, and treats gauge outputs as input triggers for the enactment of a tailored adaptation process. Our choice of process technology as the coordination paradigm for dynamic adaptation is motivated by its ability of expressing even very complex patterns of coordination and dynamic dependencies in an explicit and abstract way that is also executable and reusable. Furthermore, process enactment engines are increasingly mature, even when applied, as in agent-based systems [24] and EAI [37], to completely automated rather than human coordination subjects.

A Workflakes process unfolds according to a task decomposition strategy, which in the end generates, configures, activates groups of effectors, and coordinates them towards actuating the desired side effects onto the running target system. Effectors are considered a first-class resource: they must be explicitly described in the process enacted by the Workflakes engine.

Notice that the impact of effectors can range from the adjustment of a single operation parameter, to a method call, to complex reconfigurations of the target architecture - involving many components and connectors at once. Similarly, the technologies that can be used to implement effectors may greatly vary, depending on their reach as well as the nature of their target: they are often the most target-dependent elements in our approach, and are likely to be handcrafted. However, a significant amount of standardization of the interface between the process engine and the effectors it coordinates can be achieved, and can help decoupling the control and the actuation layers across technology and application contexts.

Such an interface is relatively simple at the conceptual level, and requires means for the process engine to *look up*, *instantiate* or *recruit*, *configure* and *activate* the effectors, and for the effectors to *report back* the outcome of their work in a meaningful form for the process engine. The implementation of this conceptual interface of course varies depending on the technological underpinnings of the effectors employed.

We have to date adopted mobile agents as our effectors, which seem particularly apt for an externalized infrastructure, since by their very nature they operate on the target system from the outside. That guarantees that new forms of adaptation computations can be easily deployed at any time onto the target with minimal disruption to service operation. In particular, the current Workflakes implementation is integrated with the Worklets mobile agent platform [19]: Worklets are code-carrying agents that Workflakes selects as effectors, configures and dispatches onto the target system, as a side effect of process steps. Each Worklet carries Java mobile code snippets, and deposits them onto one or more target components, according to a programmable trajectory. Once deposited, the execution of Worklet code is governed by constructs that specify conditional execution, repetition, timing, priority, etc. The agent transport facilities and the code execution environment are provided by Worklet Virtual Machines (WVMs) residing at all "stops" in a Worklet trajectory.

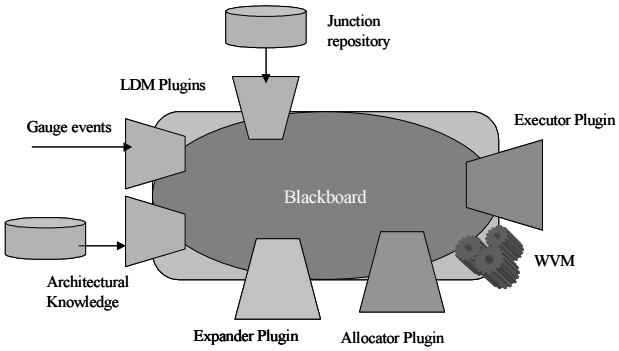

Figure 2: Representation of a typical Workflakes task processor

The current implementation of the Workflakes runtime engine relies on a specialization of the Cougaar open–source distributed platform [24]. Cougaar's decentralized task processors provide us with a number of largely autonomous controllers (as per Figure 1) for the enactment of distributed dynamic adaptation processes. Each task processor is further specified as a set of Cougaar plugins. Plugins allow to customize the functionality of task processors by inserting components that implement a particular logic or specific capability. As shown in Figure 2, a typical Workflakes task processor includes several of what Cougaar calls Logic Data Model (LDM) plugins, which are used to import and convert KX gauge events in terms of process facts, maintain internal knowledge about the target system and its state, and access a repository of effectors; an Expander plugin to load process definitions and spell them out as hierarchical decompositions of tasks; an Allocator plugin to map tasks to effectors and target components as needed; and an Executor plugin that handles the instantiation and shipment of effectors.

In Workflakes, task processors interact with Worklets in two fundamental ways. Firstly, some Worklets originate from WVMs incorporated within the task processors, and deposit their mobile code onto the target components to be adapted. Those *effectors* represent the side effects of the dynamic adaptation process onto the target system and its implementation. One of the major responsibilities of the process is therefore to decide what Worklet mobile code needs to be dispatched for a given dynamic adaptation task. That is why the repository of effectors descriptions is an essential component, and the corresponding effectors are treated as first-class process resources.

Workflakes uses Worklets also to load process definitions on the fly onto task processors, either with a pull or a push modality. The rationale for such a facility is that the dynamic adaptation of target applications is likely to call for dynamically adaptable controllers and coordinators. In Workflakes, the plugins of any task processor are initially idle and devoid of any hardcoded logic related to any particular process; for that reason, we call them *shell plugins*. The set of shell plugins launched within a task processor at start time is therefore merely indicative of the kinds of service and functionality that the cluster is meant to offer within the overall distributed Workflakes engine. Shell plugins can be activated at any time via the injection of specific *process definition Worklets*. Those Worklets dynamically deploy process fragments to the most convenient task processor for execution. Only after such deployment, shell plugins acquire a definite behavior, and start taking part in the enactment of the process. Such process delivery mechanism is effective for a centralized as well as a more scalable, decentralized process enactment architecture. It may for example be used in the pull modality for example to incrementally retrieve process fragments from a process repository when requested to handle certain specific adaptations, or in the push modality for on-the-fly process evolution across a distributed Workflakes installation.

4. Evaluation

We have carried out several experiments to validate the externalized infrastructure approach to dynamic adaptation, as embodied by KX and Workflakes

specifically. Case studies to date include such varied application domains as active networking, B2C marketplaces, Internet-wide information systems and multi-channel instant messaging.

4.1. Case study: a mass-market Internet service

Figure 3 represents the architecture of a J2EE-based multi-channel instant messaging (IM) service for personal communication, which is currently offered on a 24/7/365 basis to tens of thousands of customers through a variety of channels, such as the Web, PC-based Internet chat, Short Message Service (SMS), WAP, etc.

The service runtime environment consists of a typical three-tiered server farm: a load balancer (IBM commercial software) provides a common front end to all end-users and redirects all client traffic to several replicas of the IM components, which are installed and operate on a set of middle tier hosts. The various replicas of the IM server all share a relational database and a common runtime state repository, which make up the backend tier, and allow replicas to operate in an undifferentiated way as a collective service. Some of the IM servers are wrapped within Web applications running on commercial J2EE application servers (BEA Weblogic), others may provide additional facilities, which handle access to the service through specific channels, such as SMS or WAP, and interoperate with third-party components and resources, e.g., gateways to the cell phone communication network.

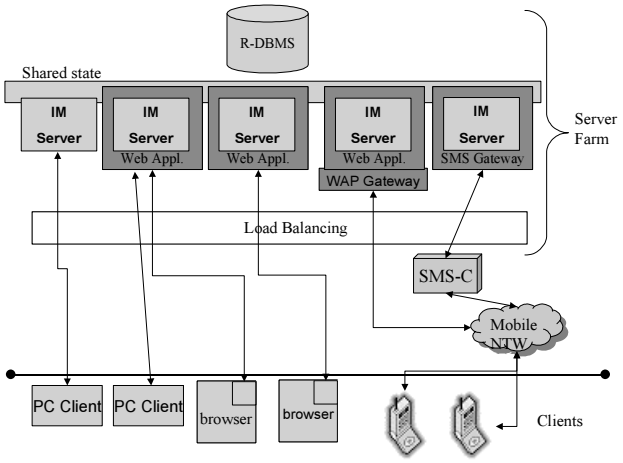

Figure 3: The IM service architecture

The case study addressed two main goals: *enhancing the QoS* perceived by end users, and *facilitating service management* by the staff in charge of supporting such a complex distributed application.

With respect to QoS, requirements focused on resolving existing load and availability problems by automating service scalability, as well as reconfiguring promptly and opportunely service parameters related to serving client requests efficiently. As for service management, requirements focused on the automation of the deployment, bootstrapping and configuration of the various service components, the continuous monitoring of those components and their interactions, and the support for "hot" service staging via automated rollout of new versions and patches.

All of those requirements are captured and addressed within a dynamic adaptation process automated by Workflakes. This process requires – among the logic and data loaded at startup onto the Workflakes engine – explicit knowledge about the service architecture and the server farm runtime environment. That knowledge is currently codified in a proprietary way.

At startup, Workflakes is given a configuration of service components that must be instantiated. Workflakes selects some hosts in the server farm for this initial deployment and sends them Worklets to execute bootstrapping code for the IM components and configure the servers with all the necessary parameters (such as the JDBC connection handle to the DBMS, the port numbers for connections by clients and other IM servers, etc.). Notice that not only the configuration information, but also the executable code of the IM server is deployed and loaded on demand, taking advantage of a code-pulling feature of the Worklets agent platform. (This approach is also followed in Software Dock [7]).

Depending on the type of the components, the deployment sub-process may change. For example, a normal IM server can be instantiated and configured by a single Worklet in one step. Web-based IM servers are notably more complex to startup and configure, since that requires first of all the spawning of a new instance of the application server, then the instantiation and parameterization of the residing Web application with respect to the hosting application server, and finally its configuration and activation as an IM component.

When a Worklet starts up an IM server, probes are simultaneously activated to track the server's instantiation and initialization. When the instantiation is successful, the process must dispatch other Worklets onto the load balancer of the server farm, which accepts traffic for the IM service, to instruct it to route it to the right host address and port for the new server. In the event of an unsuccessful initialization, instead, the likely cause is inferred by the gauge layer of the dynamic adaptation platform and reported back to the process (and also to a dashboard GUI in our implementation). Depending on the cause of that contingency, Workflakes may react in different ways: it may decide to try to bootstrap an IM server on the same host again, or on another available host, or it could skip that portion of the configuration or even abort the whole process.

Contingencies like these can be discriminated as internal or external to the process. Internal contingencies are well-known conditions that can occur in the target system and which should be provided for by specific branches of the adaptation process. External contingencies correspond either to unexpected or "new" target system conditions, or to faults within the dynamic adaptation loop itself (e.g., a communication failure between effectors and the target system components to be adapted). For those contingencies, the adaptation process should provide generic remedy strategies that are used as exceptional courses of actions, when "all else fails" or when the process needs to go back to a "safe state".

Following the initial bootstrapping phase, and after the intended service configuration is in place, Workflakes takes a reactive role, while the probing and gauging layers of the platform start monitoring and analyzing the dynamics of service usage. Certain probes and gauges are activated, to notice logging in and out of the servers, exceptions raised, service latency, number of service requests queued by the Web applications, etc.

In this case study, we are particularly concerned with load and responsiveness. Each IM server has an associated load threshold, which is best expressed in terms of the number of concurrently active clients in relationship with the memory resources of the host. When that threshold is passed, Workflakes reacts by trying to scale up the service: it selects from its system model some unused machine available in the server farm, and repeats the server bootstrapping process fragment on that machine, providing a new server replica for handling the extra load, thus achieving enhanced reliability and performance of the overall service.

For Web IM components, we are also able to reach a finer level of adaptation, exploiting the management capabilities built into the BEA Weblogic application server via Java Management eXtensions (JMX - see http://java.sun.com/products/JavaManagement), which we have integrated within the probing and effectors layers of our platform. Therefore, Workflakes can decide to intervene also in response to variations in the size of the queue of pending requests, and manipulate the details of the threading model of the Web IM application. That optimizes the degree of parallelism in processing client requests, and improves responsiveness.

We have also experimented with staging and service evolution scenarios, aiming at complete automation and minimal service disruption. It turned out that a service evolution campaign can be supported by Workflakes with relatively minor changes to the service bootstrapping process described above. The process must include tasks that gradually withdraw from the load balancer outdated server instances (thus disallowing new traffic to be assigned to them), and shut them down when traffic is absent or minimal, while another process fragment simultaneously and coordinately starts up, registers on the load balancer, and makes available to users other server instances with the new code release.

4.2 Results and lessons learned

We have been able to derive some quantitative results referring to the levels of automated support provided to the maintenance and management activities carried out onto the IM service on the field. Employing KX and Workflakes in this main case study has shown higher levels of automation, flexibility and reliability to the management of the target service and its QoS, with respect to previous labor-intensive practices. We also give some observations about the development work necessary to implement the case study.

The most significant quantitative results are:

- Reduced effort for the deployment and configuration of an IM service in the field. Current manual procedures (using Unix shell scripts and assuming DBMS and application servers pre-installed in the server farm) can take ½ to 1 person-day, with locally present experts. With KX and Workflakes, that is reduced to 1-2 minutes from a remote location.
- Reduced monitoring and maintenance effort necessary to ensure the health of the running service. A sysadmin was previously needed on-site 24/7/365, with a secondary support team of experts available on call. KX with Workflakes completely automates the monitoring of a set of major service parameters, as well as the counter-measures, for a set of well-known critical conditions.
- Reduced reaction times and improved reliability: for example, KX/Workflakes recognizes the passing of the IM load threshold in 1-2 seconds and takes approximately 40 seconds to put in place an additional server replica. Previously there was no direct overload detection: the sysadmin in charge was supposed to check the number of concurrent users from the logs and to manually start up an additional server when necessary. That was error-prone and could endanger service availability, in which case resource shortage would crash overloaded servers.
- Manageable coding complexity: KX probes, gauges and effectors are derived from generic code instrumentation templates that are then customized with situational logic. This results in rather compact code: 15 Java code lines for probes on average, usually less than 100 for effectors. In toto, the code written for the case study on top of the KX/Workflakes infrastructure was slightly above 2000 lines of Java and XML code.

Finally, other lessons we have learnt include the following qualitative considerations:

- Impact on service development: We carried out the case study positioning ourselves past the end of the development phase of the project life cycle and just prior to the deployment phase. We hence treated the target service as a complete legacy, although a legacy for which all the specifications, software artifacts and accumulated project knowledge happened to be available to the first author. Notice that also a different kind of legacy applies in the case study: the application server and the load balancer are commercial software products, which however provide sufficient APIs for carrying out our probing and actuation. Within those limitations, we were able to satisfy all the requirements of the case study.
- Relationship with architectural model: the amount of effort to analyze the target system and its behavior for dynamic adaptation purposes constituted the largest portion of the overall effort. Furthermore, a substantial portion of the software we wrote is intended to capture architectural information, relationships and inferences and represent them to KX and Workflakes. That is evidence of the strong dependency of dynamic adaptation on the ability to capture, describe and expose in an abstract and machine-readable way knowledge about the target architecture, which has motivated us to further explore integration with formal ADLs in follow-up research.
- Integrated automated management: here is where the benefit of a full-fledged process engine becomes most evident. Traditional application management is concerned with reporting warnings, alarms and other information to some knowledgeable human operator who can recognize situations as they occur, and take actions as needed. The amount of guidance and automation on the part of the management platform then may be very limited. Our approach offers instead a high level of guidance, coordination and automation to enforce what is a complex but many times largely repeatable and codifiable process.

4.3. Other case studies

Internet-scale Information Systems: the subject was ISI's Geoworlds [28], a strongly decentralized and componentized integrated Geophysical Information and Digital Library system, in experimental use for intelligence analysis at US Pacific Command (PACOM). Forms of dynamic adaptation applied to Geoworlds have varied from service parameter modification, to component repair, to global reconfigurations such as service migration.

One particularly interesting trait in this case study was that – in part building upon the lessons learned from the IM case study – we experimentally integrated architectural models and tools exploiting formal ADLs within the dynamic adaptation loop. Some of the gauges would report architecturally significant events to the CMU ABLE tool set [16], which allowed us to take dynamic adaptation decisions starting from formal architectural knowledge of the target system. That knowledge is explicit in a set of descriptions in the Acme ADL, so ABLE can decide upon and express adaptations as sets of transformations of the architectural model.

To be effected at the implementation level, transformation directives would hence be passed to Workflakes, to trigger reconfiguration processes on the deployed Geoworlds system, in accord with the architectural transformation requested. ABLE and Workflakes therefore nicely complement each other.

This juxtaposition of the architecture and implementation levels showed potential to clearly and rigorously express, reason about, validate and audit the characteristics and the effects of the modifications caused by dynamic adaptation. A difficulty we encountered and that was only partially resolved in the case study was a disconnection between the architectural model of the target system and its implementation environment; as a consequence, we have observed the need for bindings (such as those of [21]) between components and connectors in the architectural model and the runtime entities that reify the architecture in the field. Such bindings can greatly simplify the integration of ADL-based tools at all layers of our dynamic adaptation infrastructure.

The following two case studies did not directly involve the authors, but instead others applying the Workflakes system presented here:

Web Services marketplace: the subject was a prototype of an adaptive electronic marketplace. Said marketplace interfaces with a number of service components implemented as Web Services by multiple providers, and offers complex, composite service chains [27]. The dynamic adaptation platform keeps under control the basic functioning parameters of participating Web Services (such as availability, responsiveness, transaction completion ratio, etc.), analyzes their accumulated performance, and uses this information to adapt the behavior of the mediator component of the marketplace, in particular the mechanisms used in selecting service providers for composing service offers to the customers' satisfaction.

Active networking: the subject was the dynamic adaptation of active network elements, in particular active firewalls, which can be reconfigured on the fly in response to network conditions, the kind of traffic they receive, users' profiles, etc. [29]. The case study used Workflakes

process-based adaptation coordination in conjunction with different implementations of the probing, gauging and effectors layers, based on TTCN-3 [36]. In response to an analysis of the network packets arriving at the firewalls and of their filtering performance and criteria, Workflakes would replace the code installed within active firewall nodes, effectively modifying their behavior. The dynamic adaptation process also includes provisions for the validation of newly installed firewall configurations through their on-line testing.

5. Related work

Given the focus of the paper and space limitations, we only compare here Workflakes in relation to other work that apparently exploits process technology to control the behavior and performance of a running application. We do not expand to comparing KX as a whole, or any of its other various constituents, or even the underlying externalized infrastructure model, to the many other approaches addressing the problem space of dynamic adaptation in whole or in part.

However, we notice that it is the externalized stance that most strongly characterizes Workflakes. Often, in fact, automated solutions to software coordination and control present structural dependencies with respect to the subjects of their coordination.

Some of those solutions can be seen as an evolution of built-in fault tolerance code. For example, [15] proposes a rule-based inference engine for decision support in application-level QoS assurance, including a coordination entity guiding a set of computational actuators. However, the coordinator and actuators must both be embedded with each target component. That makes it more difficult to define system-wide adaptations and limits the adaptations that can be carried out without rebuilding the target.

Another classic approach is that of an environment or middleware with native dynamic adaptation capabilities. Generic (i.e., not necessarily process-based) examples of dynamic adaptation middleware include Conic [14], Polylith [12], 2K / dynamicTao [13] and many others; they all offer a set of dynamic adaptation primitives as a premium for applications built with and operating on top of themselves.

Also many works that employ process technology for software control and coordination adopt in fact a middleware-like approach, by exerting the coordination "from the inside", that is, on the target's own computations. For example, [10] introduces Containment Units, as modular process-based lexical constructs for defining how distributed applications may handle self-repair and self-reconfiguration. Containment Units define a hierarchy of processes that predicate on constraints and faults, and take action to handle faults within the defined constraints. The enactment of Containment Units is under the responsibility of a process engine that is integral to the system being adapted, and proceeds by directing changes on the target components, which by definition is process-aware.

PIE [8] is another example of a process-based middleware, which supports federations of components. PIE adds a control layer on top of a range of inter-component communication facilities. The control layer implements process guidance via handlers that react to and manipulate the communications exchanged by the components in a federation. Dynamic adaptation is thus limited to the reconfiguration of the service architectural connectors and is carried out by plugging in appropriate handlers, as directed by the process, which intrude in the normal course of computation of the target.

TCCS [9] has considerable similarities with Workflakes, since it employs its process engine to direct the work of analogous effector agents, to carry out the dynamic adaptation tasks. However, TCCS is the epitome of the middleware approach, since it is in charge of all interactions between the system components, even normal operations; that is, the target application simply does not exist independently from its process and agent-based framework.

With each of these dynamic adaptation middlewares, all service components need to be assembled from the start according to the middleware and its primitives. This not only poses a considerable barrier with respect to legacy software, but also introduces a very strong dependency between actors and subjects of dynamic adaptation. Furthermore, the spectrum and granularity of possible adaptations is effectively restricted by the set of primitives made available by the chosen middleware. A similar observation applies also to those works that exploit the characteristics of established computing frameworks to facilitate certain aspects of dynamic adaptation, such as BARK [11], which is limited to the EJB component model.

In contrast to all of the above, Workflakes remains independent from any underlying computing framework and quite general with respect to the reach, granularity and kinds of dynamic adaptation that it can exert, since the target is fully disjoint from the dynamic adaptation engine.

The most similar approach (that we know of) may be Willow [17]. Willow proposes an architecture for the survivability of distributed applications, analogous to our vision of a superimposed feedback loop. In particular, Willow can implement reactive as well as proactive dynamic adaptation policies, which are driven by codified architectural knowledge, and enacted via a process-based mechanism built upon the previous Software Dock

(re)deployment engine [7]. It appears, however, that Willow restricts itself to coarse-grained reconfigurations, such as replacing, adding and removing entire components, perhaps even composite substructures, from the target application, while presuming conventional embedded approaches for more local and refined adaptations.

6. Contributions and forecast

Workflakes is among the results of a multi-institution consortium effort concerned with instrumenting, measuring and controlling pre-existing distributed software systems. We describe/reference elsewhere our own/others' approaches to instrumenting and measuring, and note that the consortium has researched "standard" interfaces for these components to communicate with each other as well as with the control component. Workflakes is the first control component that has been developed within the consortium.

Workflakes has adequately demonstrated the process-based software adaptation controller concept in the case studies covered here. In particular, while the instrumentation and measurement facilities by themselves may reduce a feedback loop that was previously entirely manual from days or weeks to hours, Workflakes has shown that an automated controller can further reduce the consequently determined software adaptations from hours to minutes and seconds. Of course, not all adaptations can be fully automated, so our continuing research will try to better characterize those that can vs. cannot.

To investigate further the boundaries of our approach and its usability, and assess its merits as well as its drawbacks, we are also working on applying Workflakes to particularly demanding domains. One such domain is that of systems with (soft) real-time adaptation requirements, e.g., AI2TV – Adaptive Internet Interactive Team Video – where we employ KX and Workflakes to keep "in sync" multiple viewers of the same distance learning lecture video at varying compression levels, see `http://www.psl.cs.columbia.edu/ai2tv`). We compare and contrast the performance of an externalized adaptation loop – with its induced reaction latency – against middleware-oriented approaches to similar problems, such as [32] [34]. We are similarly interested to weigh the performance of Workflakes in those cases against other externalized infrastructures that employ for the coordination of actuation means different from process technology, such as [33].

Another investigation issue is how the reliability of externalized dynamic adaptation facilities may influence that of the target system, and how to minimize any adverse impacts deriving from problems and failures that may occur at the various stage of the adaptation loop.

We also plan to precisely define and develop standard interfaces for Workflakes in its role as the control component of KX, similarly to what has been done for the other infrastructure layers. That includes adopting a common, high-level process formalism suitable for describing and enacting dynamic adaptation processes (we are currently evaluating the UMass Little-JIL process language [6]), expanding the repertoire of effectors beyond mobile agents (we are currently integrating SOAP-based effectors), and being able to handle all effectors in a uniform way, independently from their technology.

Finally, another research thread is to more tightly integrate Workflakes with architecture-based modeling and analysis tools, leading to selection/construction of architecture-based repair strategies (e.g., [31] [16]), which are then spelled out and implemented as processes.

7. Acknowledgements

We would like to thank Gaurav Kc for his ongoing development of Worklets, Lee Osterweil and Nathan Combs for the frequent discussions and suggestions about Workflakes, George Heineman for help with techniques for KX and Worklets, Bob Balzer, David Garlan, Bradley Schmerl, David Wells and David Wile for their insights on the general infrastructure model and APIs' standardization, Pier Giorgio Bosco, Mario Costamagna, Matteo Demichelis, Elio Paschetta and Roberto Squarotti at TILAB for their contribution on applicability and to the IM service case study, the partner organizations and all the colleagues who worked in the OLIVES project, and the other members of the Programming Systems Lab. PSL is funded in part by Defense Advanced Research Project Agency under DARPA Order K503 monitored by Air Force Research Laboratory F30602-00-2-0611, by National Science Foundation CCR-9970790, EIA-0071954 and CCR-0203876, by Microsoft Research and by IBM. The work at TILAB is funded in part by EURESCOM project P-1108 (OLIVES).

8. References

[1] IBM Research, "Autonomic Computing Manifesto", http://researchweb.watson.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[2] A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies".

UC Berkeley Computer Science Technical Report UCB//CSD-02-1175, March 15, 2002.

[3] J. Salasin, "Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA)", http://www.darpa.mil/ito/research/dasada/.

[4] B. Balzer, "Probe Run-Time Infrastructure", http://www.schafercorp-ballston.com/dasada/2001WinterPI/ProbeRun-TimeInfrastructureDesign.ppt

[5] G. Valetto, G. Kaiser, and G.S. Kc, "A Mobile Agent Approach to Process-based Dynamic Adaptation of Complex Software Systems", in *8th European Workshop on Software Process Technology*, June 2001. http://www.psl.cs.columbia.edu/ftp/psl/CUCS-001-01.pdf.

[6] A.G. Cass, B. Staudt Lerner, E.K. McCall, L. J. Osterweil, S.M. Sutton, Jr., and A. Wise, "Little-JIL/Juliette: A Process Definition Language and Interpreter", in *22nd International Conference on Software Engineering*, June 2000.

[7] R.S. Hall, D. Heimbigner, and A.L. Wolf, "A Cooperative Approach to Support Software Deployment Using the Software Dock", in *21st International Conference on Software Engineering,* May 1999.

[8] G. Cugola., P.Y. Cunin, S. Dami, J. Estublier, A. Fuggetta, F. Pacull, M. Riviere, and H. Verjus, "Support for Software Federations: The Pie Platform," in *7th European Workshop on Software Process Technology*, February 2000.

[9] S.K Shirvastava, L. Bellissard, D. Feliot, M. Herrmann, N. De Palma, and S.M. Wheater, "A Workflow and Agent based Platform for Service Provisioning", in *4th IEEE/OMG International Enterprise Distributed Object Computing Conference*, September 2000

[10] J.M. Cobleigh, L.J. Osterweil, A. Wise, and B. Staudt Lerner, "Containment Units: A Hierarchically Composable Architecture for Adaptive Systems", in *10th International Symposium on the Foundations of Software Engineering (FSE 10)*, Charleston, SC, November 2002.

[11] M.J. Rutherford, K. Anderson, A. Carzaniga, D. Heimbigner, and A.L. Wolf, "Reconfiguration in the Enterprise JavaBean Component Model", in *IFIP/ACM Working Conference on Component Deployment*, June 2002.

[12] C.R. Hofmeister, and J.M. Purtilo, "Dynamic Reconfiguration in Distributed Systems: Adapting Software Modules for Replacement", in *13th International Conference on Distributed Computing Systems*, May 1993.

[13] F. Kon, R. Campbell, M.D. Mickunas, K. Nahrstedt, and F.J. Ballesteros. "2K, A Distributed Operating System for Dynamic Heterogeneous Environments", in *9th IEEE International Symposium on High Performance Distributed Computing*, August 2000.

[14] J. Magee, J. Kramer, and M. Sloman. "Constructing Distributed Systems in Conic*", IEEE Transactions on Software Engineering*, 15(6):663--675, June 1989.

[15] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer, "Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework", in *3rd IEEE International Workshop on Policies for Distributed Systems and Networks*, January 2001.

[16] B. Schmerl, and D. Garlan, "Exploiting Architectural Design Knowledge to Support Self-repairing Systems", in *14th International Conference on Software Engineering and Knowledge Engineering*, July 2002.

[17] J.Knight, D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, and P. Devanbum, "The Willow Survivability Architecture", in *4th Information Survivability Workshop (ISW-2001)*, Vancouver, B.C.,18-20 March 2002.

[18] P.N. Gross, S. Gupta, G. E. Kaiser, G.S. Kc, and J.J. Parekh, "An Active Events Model for Systems Monitoring", in *Working Conference on Complex and Dynamic Systems Architecture*, December 2001. http://www.psl.cs.columbia.edu/ftp/psl/CUCS-011-01.pdf.

[19] G. Kaiser, A. Stone, and S. Dossick, "A Mobile Agent Approach to Lightweight Process Workflow," in *International Process Technology Workshop*, September 1999. http://www.psl.cs.columbia.edu/ftp/psl/CUCS-021-99.pdf.

[20] D. Garlan, B. Schmerl, and J. Chang, "Using Gauges for Architecture-Based Monitoring and Adaptation", in *Working Conference on Complex and Dynamic Systems Architecture*, December 2001.

[21] E.M. Dashofy, A. van der Hoek, and R.N. Taylor, "An Infrastructure for the Rapid Development of XML-based Architecture Description Languages", in the *24th International Conference on Software Engineering*, May 2002.

[22] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimbinger, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software", *IEEE Intelligent Systems* 14(3): 54-62, May/June 1999.

[23] G. Kaiser, P. Gross, G. Kc, J. Parekh, and G. Valetto, "An Approach to Autonomizing Legacy Systems", in *Workshop on Self-Healing, Adaptive and Self-MANaged Systems (SHAMAN 2002),* June 2002.

[24] Cougaar Home Page, "Welcome to the Cognitive Agent Architecture (Cougaar) Open Source Website". http://www.cougaar.org.

[25] Gail Kaiser, "Autonomizing Legacy Systems", invited talk at the Almaden Institute Symposium on Autonomic

Computing, April 10-12 2002 http://www.almaden.ibm.com/institute/pdf/KaiserGail.pdf

[26] Giuseppe Valetto, and Gail Kaiser, "A Case Study in Software Adaptation", in *Workshop on Self-Healing Systems (WOSS '02)*, 18-19 November 2002.

[27] A. Rocha, G. Valetto, E. Paschetta, and S. Heikkinen, "Continuous On-Line Validation of Web Services", in *International Conference on Electronic Publishing (ELPUB 2002)*, November 6-8, 2002.

[28] M. Coutinho, R. Neches, A. Bugacov, V. Kumar, K. Yao, I. Ko, R. Eleish, and S. Abhinka, "GeoWorlds: A Geographically Based Information System for Situation Understanding and Management", in *1st International Workshop on TeleGeoProcessing (TeleGeo 99)*, Lyon, France, May 6-7, 1999.

[29] P. Deussen, G. Valetto, G. Din, T. Kivimaki, S. Heikkinen, and A. Rocha, "Continuous On-Line Validation for Optimized Service Management" in *EURESCOM Summit 2002*, Hiedelberg, Germnany, October 21-24, 2002.

[30] E. Kasten, P. K. McKinley, S. Sadjadi, and R. Stirewalt, "Separating introspection and intercession in metamorphic distributed systems", in *IEEE Workshop on Aspect-Oriented Programming for Distributed Computing*, July 2002.

[31] Carnegie Mellon University, "Acme Web, The Acme Architectural Description Language", http://www-2.cs.cmu.edu/~acme/.

[32] C. Poellabauer, H. Abbasi, and K. Schwan, "Cooperative Run-time Management of Adaptive Applications and Distributed Resources", in *ACM Multimedia*, Juan-les-Pins, France, December 1-6, 2002.

[33] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. "On Adaptive Resource Allocation for Complex Real-Time Applications", in *18th IEEE Real-Time Systems Symposium*, December 1997.

[34] R. Vanegas, J.A. Zinky, J.P. Loyall, D.A. Karr, R.E. Schantz, D.E. Bakken, "QuO's Runtime Support for Quality of Service in Distributed Objects", in *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, 15-18 September 1998, The Lake District, England.

[35] W. Gu, G. Eisenhauer, and K. Schwan, "Falcon: On-line Monitoring and Steering of Parallel Programs", in *Concurrency: Practice and Experience*, 10(9): 699-736, August 1988.

[36] "Tree and Tabular Combined Notation, version 3," ITU-T Recommendation Z.140, 2001 http://www.itu.int/ITU-T/studygroups/com10/languages/Z.140_0701_pre.pdf

[37] F. Cubrera, Y. Goland, J. Klein, F. Leymann, D. Roter, S. Tatte, and S. Weerawarana, "Business Process Execution Language for Web Services, Version 1.0", July 2002, http://www.ibm.com/developerworks/library/ws-bpel/