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 applications. This feedback-loop infrastructure is primarily targeted to pre-existing systems and thus operates outside the application itself without making assumptions about the target system's internal communication/computation mechanisms, implementation language/framework, availability of source code, 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 actuators or effectors somehow attached to the target system, so we briefly sketch one such facility (elaborated elsewhere). The core of the paper is the model, architecture, and implementation of Workflakes*, the decentralized process engine we use to tailor, control, coordinate, respond to contingencies, etc. regarding a cohort of such actuators. We have validated our approach and the* Workflakes *prototype in several case studies, related to different application domains. Due to space restrictions we concentrate primarily on one case study, elaborate with some detais a second, and only sketch others.*

1. Introduction

Distributed computing is becoming a commodity. Users rely upon distributed systems for a number of value-added services that pervade their everyday lives, such as Web-based collaboration suites, electronic B2B and B2C, on-demand multimedia content provisioning, ubiquitous personal messaging, and many others, which are built on top of a networking infrastructure as distributed systems, often constructed by composition. The complexity of the behavior and interrelationships of these "systems of systems" becomes increasingly harder to analyze in advance, and keep under control on 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 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 bringing it down. This comports the presence of facilities for the *dynamic adaptation* of complex, distributed software systems and services. By that term we mean any automated and controlled 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. Example may range from tuning functioning parameters in order to optimize performance, to architecture-wide interventions such as service deployment or service migration.

The Dynamic adaptation theme is gaining attention as an opportunity to address the ever-increasing complexity of IT infrastructures and applications. For example, the autonomic computing initiative announced by IBM [1], or the Recovery Oriented Computing works at Berkeley and Stanford [2] go in that direction. But it also constitutes a challenge, since it can be seen as a form of automated maintenance of "live" systems. As such, it tends to be even more complex than conventional off-line maintenance.

The hardwiring of self-adaptation provisions in 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 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 that remain orthogonal to the target system's main computation, control and communication,

constituting an *externalized* dynamic adaptation infrastructure. This approach enables retrofitting with the desired reconfiguration, self-healing, and self-management capabilities also legacy systems and systems built by composition with third-party components.

Our model for externalized dynamic adaptation is that of a layered architecture, which comprises layers for data collection, information analysis, decision / control, and actuation. In-depth discussion of the overall infrastructure model, and how its collection and analysis components have been fulfilled within our *Kinesthetics eXtreme (KX)* implementation can be found in [18] [23]. This paper focuses instead on the decision and control role, and how we have addressed it in KX with our *Workflakes* process-based engine. Workflakes employs process technology to coordinate the actuation layer, which is currently provided via our worklets mobile agents effectors [5] [19]. A preliminary paper [5] sketched the ideas at the basis of the Workflakes project. This paper provides the first complete presentation of the Workflakes model and architecture, according to our recently completed operational implementation.

KX and Workflakes have been evaluated in a number of case studies. We present here in full detail one such case study on an industrial Internet service with its most recent results, which subsume and update a previous report [26]. We also sum up other case studies in their most interesting traits.

2. A model for 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 existing distributed systems for the purposes of continually monitoring and modifying their 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. That in turn can be achieved only via interoperability and standardization of the interactions between the infrastructure components, so that the numerous technological options that can be used to implement probes, gauges, controllers and effectors can be easily accommodated by the model.

As a consequence, within the DASADA program [3], under which we are conducting this research, a standard model, the Common DASADA Infrastructure (CDI) [25] has emerged, which organizes the design of the feedback loop according to multiple layers, as shown in Figure 1.

In the first place, the Collection layer 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, against some models that characterize the target system, and findings are reported to the Gauge Bus. At that point, 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 onto it. Adaptation actions would be carried by effectors at the Actuation layer, under the coordination of one or more controllers. Effectors would actuate (i.e. reconfigure, tune or otherwise adapt) individual components, as well as connectors and major substructures of the system.

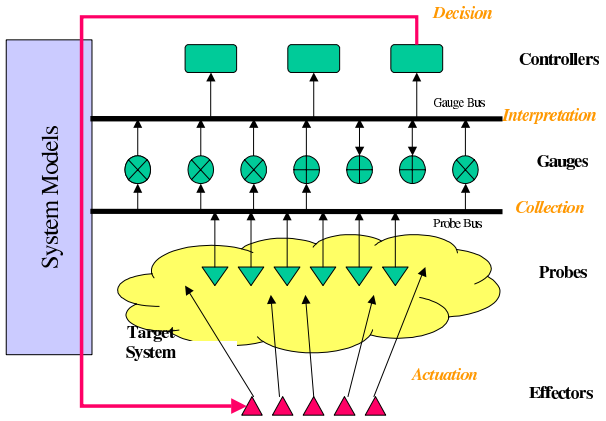

Figure 1: Dynamic adaptation infrastructure.

Notice that while this model is largely independent of the running target, this is not to say that the specific probes, gauges, controllers, effectors and the models that govern them are themselves independent of the running 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 with makes easier to separate 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 CDI, much work has been devoted to the development of standard probing [4], as well as a gauging [20] APIs. Standards for the decision and control roles in a dynamic adaptation platform are however less well understood.

To close the loop, the infrastructure must also automate decisions on the adaptation to be carried out, control the adaptation as it occurs on the target system, 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 exert it on multiple effectors taking part in the actuator layer.

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, gauges 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, coupled with constraint analysis and architecture transformation tools, such as in [16] [22].

When a decision upon some adaptation is taken, a single action will sometimes suffice to fulfill it. In most cases, however, the decision will have to be mapped onto several finer-grained activities, impacting various implementation elements. In the latter case, a sophisticated coordination mechanism is needed: some of those activities may be conditional, or dependent on others, or may fail, calling for contingency planning.

To address that complexity, Workflakes relies on process-based coordination, and treats decisions as triggers for an adaptation process. The process unfolds according to a task decomposition strategy that in the end generates, configures, activates sets of effectors, and coordinates them towards actuating the desired side effects onto the running target system. Effectors are thus considered a first-class resource in Workflakes: they must be explicitly described in the process and made available to 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 connector at once. Similarly, the technologies that can be used for 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. Standardization of the interface between the process engine and effectors is another objective of the CDI work in the next future.

We have to date adopted mobile agents as our effectors, since they operate by their very nature on the target system form the outside, guaranteeing 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 are selected as effectors, configured and dispatched as a side effect of process steps. Each worklet carries Java mobile code snippets, named junctions, which are deposited onto one or more target components, according to a trajectory that can be programmed. Once deposited, the execution of code in a junction is governed by its encasing jacket, a construct that specifies conditional execution, repetition, timing, priority, etc. The agent transport facilities and execution environment are provided by Worklet Virtual Machines (WVMs) residing at all "stops" in a worklet trajectory.

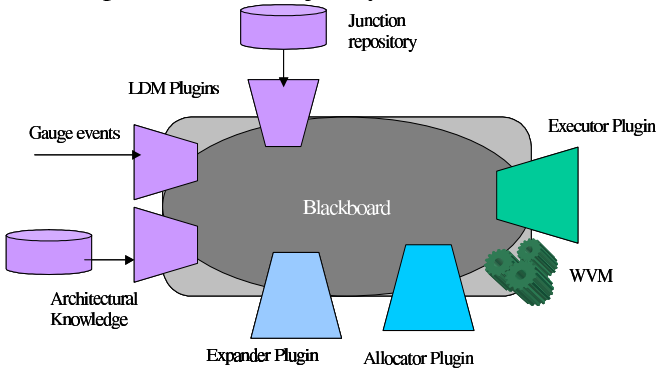

Figure 2: Representation of a typical Workflakes task processor.

The Workflakes process runtime engine in its current implementation relies on a specialization of the Cougaar open–source system [24]. Cougaar decentralized task processors (or clusters) provide us with a number of largely autonomous controllers (as per

Figure 1) for the enactment of the dynamic adaptation process. Each cluster 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 a specific capability. As shown in Figure 2, a typical Workflakes task processor includes several Logic Data Model (LDM) plugins 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 worklet junctions effectors; an Expander plugin to load process definitions and spell them out as hierarchical decompositions of tasks; an Allocator plugin to map tasks to junctions and to target components as needed; an Executor plugin that handles the instantiation and shipment of effectors.

In Workflakes, task processors and plugins are specifically constructed to operate with worklets, in two fundamental ways. First, worklet originate from WVMs incorporated within the task processors, and deposit junctions onto the target components to be adapted. One of the major responsibilities of the process is therefore to decide what *effectors junctions* need to be dispatched, for a given dynamic adaptation task. That is why the

repository of junction descriptions, is an essential component, since effectors must be treated as first-class process resources.
Furthermore, Workflakes uses worklets also to dynamically load process definitions onto task processors, either with a pull or a push modality. In Workflakes, plugins 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 cluster at start time is therefore merely indicative of the kinds of service and functionality that the cluster is meant to offer within the overall Workflakes engine. Shell plugins can be activated at any time via the injection of specific *process definition junctions*. Those junctions 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 may for example be used in the pull modality to incrementally retrieve process fragments when requested to handle certain adaptations, or in the push modality for on-the-fly process evolution across a distributed Workflakes installation.

4. Evaluation

A number of case studies have been carried out to experiment with and validate the externalized infrastructure approach to dynamic adaptation we have discussed, 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 multi-channel instant messaging (IM) service for personal communication we have been experimenting with. That J2EE-based service is currently offered on a 24/7/365 basis to thousands of users 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 (an 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 back end tier, and allow replicas to operate in an undifferentiated way as a collective service. Some of the IM servers are wrapped within Web application running on J2EE application servers (BEA Weblogic), other 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.

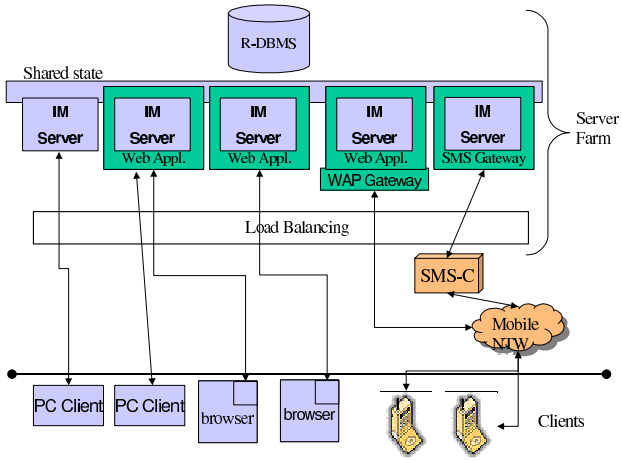

Figure 3: The IM service architecture.

We have organized the case study according to several iterations. Each iteration tries to fulfill a larger set of adaptation requirements originating from the service provisioning organization, and elicited from the application development and maintenance team. At the same time it extends the reach of the dynamic adaptation platform to a larger subset of the IM service components.

We have now completed two iterations. In the first iteration, we applied dynamic adaptation only to "bare" IM servers, and we addressed principally relatively coarse-grained adaptations. In the second iteration, we have encompassed the application servers and the residing Web applications as dynamic adaptation targets, and addressed finer-grained adaptations. In the next iteration, we plan to cover the SMS component, which has in this meanwhile become a critical channel for the service traffic, with its specific reliability and availability issues.

Up to now, the case study has 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 and (in the second iteration) reconfiguring promptly and opportunely service parameters related to serving client requests efficiently. As for service management, requirements focused on the automated deployment, bootstrapping and configuration

of the various service components, as well as continuous monitoring of those components and their interactions, and (in the second iteration) 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 that is automated by Workflakes and whose side effects on the implemented system are caused by worklets. 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. The configuration must include at least one "normal" IM server and any number of Web-based IM servers. Workflakes selects some hosts in the server farm for this initial deployment and sends worklets to them. Those worklets execute bootstrapping code for the IM components and configure them 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 from a code repository made available to the incoming worklet, 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 start-up 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 the worklets starts up an IM server, probes are simultaneously activated to track its instantiation and initialization. When the instantiation is successful, the process must dispatch other worklets onto the load balancer of the server farm, to instruct it to accept traffic for the IM service and 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 to a dashboard GUI, as well as to the process. Depending on the cause of that contingency, Workflakes may react in different ways: for example, it may decide to try to bootstrap an IM server on the same machine again, or on another available one, or it could skip that portion of the configuration, or even abort the whole process.

More in general, contingencies like that 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 (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, such as 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 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 trying to scale the service up. It selects some unused machine that is still 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, and 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 in the BEA Weblogic application server via Java Management eXtensions (JMX - see http://java.sun.com/products/JavaManagement), which we have integrated during the second iteration 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.

In the second iteration, 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 old 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 by the KX to the maintenance and management activities carried out onto the IM service on the field. Employing KX and Workflakes in the case studies has shown higher levels of automation, flexibility and reliability to the management of the target service and its QoS, with respect to previous labour-intensive practices. We also provide observations about the development work necessary to implement the case study (on top of the platform implementation offered by KX). The most significant quantitative results are:

- Reduced effort for the deployment and configuration of an IM service on 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, 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 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 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 case resource shortage would crash overloaded servers.
- Manageable coding complexity: KX probes, gauges, effectors junctions are derived from generic code instrumentation templates that are then customized with of situational logic. This results in rather compact code: 15 Java code lines for probes on average, usually less than 100 for effectors. In total, the code written for the case study 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 whole 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 us. Notice that also a different kind of legacy takes part in the case study: the application server and the load balancer are commercial software products, by BEA and IBM, 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 Workflakes and KX. That suggests a strong dependency of dynamic adaptation on the ability to capture, describe and expose in an abstract and machine-readable way the knowledge about the target architecture, which has motivated us to 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 if needed. The amount of guidance and automation on part of the management platform is 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-wide Information Systems: the subject was ISI's Geoworlds [28], a strongly decentralized and componentized integrated Geographical Information and Digital Library system, in 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 on the lessons learnt in the IM case study – we experimented integrating architectural models

and tools exploiting formal ADLs within the dynamic adaptation loop. Some of the gauges would report architecturally significant events to the ABLE tool set [16], which allowed us to take dynamic adaptation decisions starting from an architectural knowledge of the target system. That knowledge is captured in a set of descriptions in the Acme ADL, and ABLE is able to decide upon and express adaptations as sets of changes to 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 could therefore nicely complement each other.

This juxtaposition of the architecture- and implementation levels – although preliminary - showed potential to offer means 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 and the implementation environment; as a consequence, we have observed that it is necessary to have bindings (such as those of [21]) between components and connectors in the architectural model and the runtime entities that reify the architecture on the field. Such bindings can greatly simplify the integration of ADL-based tools at all layers of our dynamic adaptation infrastructure.

Web Services marketplace: the subject was the validation of the performance and the offer of an electronic marketplace, which interfaces with a number of service provider components implemented as Web Services [27]. The dynamic adaptation platform would keep under control the basic functioning parameters of participating Web Services (such as availability, responsiveness, transaction completion ratio, etc.), analyze their accumulated performance, and use this information to adapt the behavior of the mediator components 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 that 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 in conjunction with different implementations of the probing, gauging and effectors layers. Workflakes would replace the code installed within active firewall nodes, in response to an analysis of the network packets arriving at the firewalls and of their filtering performance and criteria. The dynamic adaptation process also included provisions for the validation of installed firewall configurations through their on-line testing.

5. Related work

Given the focus of the paper and space limitations, we only discuss here Workflakes in relation to other works that propose to exploit process technology to control the behavior and performance of a running application, rather than comparing KX or even the 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 more difficult to define system-wide adaptations and limits the adaptations that can be carried out without re-building the target.

Another classic approach is that of an environment or middleware with native dynamic adaptation capabilities. Generic (i.e. non 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 of the works that use process technology for SW 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 need to be process-aware to some degree.

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, intrudes 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 effectors 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 service components, even normal operations; that is, the target services simply do not exist independently from its process and agent-based framework.

With every dynamic adaptation middleware, all service components would 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 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 that of 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 sub-structures, from the target application, while presuming conventional embedded approaches for more local and refined adaptations.

6. Contributions and forecast

Workflakes is part 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 is developing "standard" interfaces for these components to communicate with each other as well as with the control component. Workflakes is the first sample control component that has been developed within the consortium, acting as a "proof of concept". The consortium plans to later develop standard interfaces for such control components, including possibly adopting a common process language, as well as implementing and experimenting with other controller examples.

Workflakes has adequately demonstrated the software adaptation controller concept in the case studies covered here. In particular, the customer for the IM service study will make the final decision on going into production use only two days after the deadline for this paper submission. While the instrumentation and measurement facilities by themselves may reduce the previously manual feedback loop 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.

The immediate next step is to more fully integrate Workflakes with feedback-loop applications involving architecture-based modeling and analysis, leading to selection/construction of architecture-based repair strategies (e.g., [31] [16]). Other near-future work includes integration with the Little-JIL process language [6], as well as continuation of the GeoWorlds case study and (hopefully) soon application of our work to other real-world systems representing other domains, such as command and control.

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 the IM service case study, the other members of the Programming Systems Lab for their work on KX. The consortium referred to in the paper currently consists of BBN, CMU, Columbia, OBJS, Teknowledge, University of Colorado, UMass, and WPI. 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 and EIA-0071954, and by Microsoft Research. 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] Balzer, B., "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 the *10th International Symposium on the Foundations of Software Engineering (FSE 10)*, Charleston, SC, November 2002. To appear.

[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", Columbia University Department of Computer Science, TR # CUCS-019-02, July 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. To appear.

[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. To appear.

[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/.