LIBERO: A Framework for Autonomic Management of Multiple Non-functional Concerns*

Marco Aldinucci[1], Marco Danelutto[2], Peter Kilpatrick[3], and Vamis Xhagjika[2]

[1] University of Torino
[2] University of Pisa
[3] Queen's University Belfast

Abstract. We describe a lightweight prototype framework (LIBERO) designed for experimentation with *behavioural skeletons*—components implementing a well-known parallelism exploitation pattern *and* a rule-based autonomic manager taking care of some non-functional feature related to pattern computation. LIBERO supports multiple autonomic managers within the same behavioural skeleton, each taking care of a different non-functional concern. We introduce LIBERO–built on plain Java and JBoss–and discuss how multiple managers may be coordinated to achieve a common goal using a two-phase coordination protocol developed in earlier work. We present experimental results that demonstrate how the prototype may be used to investigate autonomic management of multiple, independent concerns.

Keywords: structured parallel/distributed programming, behavioural skeletons, non-functional concerns, performance, security, autonomic management.

1 Introduction

A behavioural skeleton (BS) is the result of the co-design of a well-known, efficient parallelism exploitation pattern *and* of a rule-based control loop implementing an autonomic manager of (one or more) non-functional properties related to the pattern [1,2]. The concept was introduced to tackle the problem of efficient, autonomic management of non-functional features of parallel/distributed computations, such as performance, security, fault tolerance, power management, etc. The BS parallel pattern makes use of well-understood techniques to implement that particular pattern on target architectures. The BS autonomic manager executes a classical Monitor, Analyse, Plan, Execute (MAPE) control loop to monitor and adjust the system to modify some non-functional characteristics.

Behavioural skeletons were originally designed in the framework of GCM, the Grid Component Model [8] developed within CoreGRID [7] and subsequently

* This work has been partially supported by ERCIM/CoreGRID.

M.R. Guarracino et al. (Eds.): Euro-Par 2010 Workshops, LNCS 6586, pp. 237–245, 2011.

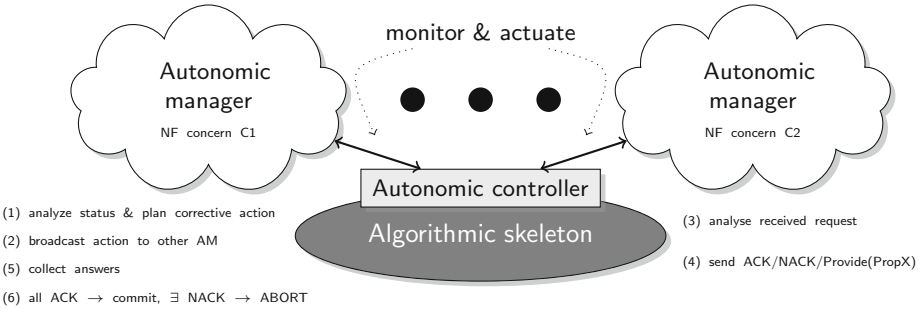


Fig. 1. Coordinating activities of distinct autonomic managers in a BS

implemented in the GCM reference implementation built on top of ProActive [14] in GridCOMP [11]. GridCOMP produced a GCM BS prototype supporting common stream parallel patterns–pipelines and farms–with managers taking care of performance issues. Those BS were demonstrated to be effective in managing (best-effort) user-supplied *performance contracts*. In [2] it was shown how contracts requiring a given throughput can be guaranteed when a single BS models the entire application. In [3] we introduced techniques that support the coordination of the different managers in a BS hierarchy.

In the general case, however, *multiple* non-functional concerns have to be addressed within the same computation. The BS concept can be easily extended in such a way that multiple managers are associated with the same parallel pattern, each taking care of a different concern. In [4] we identified the need for such managers to interact to achieve consensus before effecting changes to the managed system. We also identified protocols for achieving such consensus. However, no actual implementation was presented. In this paper we introduce LIBERO (LIghtweight BEhaviouRal skeletOn framework), which is a lightweight prototype implementing several BS–including pipes and farms–and supporting multiple autonomic managers within a single BS.

2 Autonomic Manager Coordination

Problems may arise when *independent* autonomic managers are run within the same behavioural skeleton. In a scenario such as that depicted in Fig. 1, multiple managers are associated with the same algorithmic skeleton in a single BS. The algorithmic skeleton implements a well-know parallelism exploitation pattern. Through its *autonomic controller* (AC) it provides i) methods to access its internal state (to support monitoring) and ii) methods to operate on its internal state (to modify its behavior). Each associated autonomic manager takes care of a distinct non-functional concern.

When different AMs associated with the same BS independently decide to take some action those actions must be coordinated as they may produce effects that are mutually incompatible. In [4] we introduced a two-phase approach to

coordinate different manager activities. In this approach each action planned by an AM is validated by the other AMs in the BS before being executed. The manager taking care of non-functional concern X (e.g. performance), analyzes system behaviour and decides to take some action (① in Fig. 1). It informs the other managers of the decision ②. These managers evaluate ③ the decision with respect to any consequences for their non-functional concern. Eventually they return ④ one of three answers: `ACK`, meaning the decision can be safely taken by the first manager, `NACK`, meaning the decision is in conflict with the managed non-functional concern and therefore should be aborted, or `provide(property)`, meaning the decision may be actuated provided `property` is ensured (e.g. securing of connections). The manager initiating the process gets answers from the other managers ⑤ and either actuates its decision (the original plan or a modified one to accomplish `property`) or aborts it ⑥.

This two-phase protocol has not previously been experimented with, due mainly to the difficulty of embedding a complex management structure in the reference implementation of BS in ProActive/GCM. We implemented LIBERO to allow assessment of the feasibility of this protocol as well as to experiment with other protocols regulating autonomic management.

3 LIBERO

LIBERO is a prototype supporting BS with multiple autonomic managers implemented using lightweight components. Each component implementing a parallel computation has a managing entity–the AM–that deals with the *non-functional* aspects of the parallel computation in a local and autonomic way. The AM management functions operate through the operations provided by the component Autonomic Controller [8]–the AC–which exports its internal computation state and provides a set of operations to modify component state and functioning.

LIBERO implements the BS previously investigated in GridCOMP, namely those modelling the usual stream parallel patterns, such as *task farm*s and *pipeline*s [6], and equipped with a single autonomic manager taking care of a single non-functional concern. In addition, LIBERO supports *Multiple Concern Management*, implementing the coordination algorithm outlined in Sec. 2. All LIBERO components are native *Java* objects. This simplifies investigation of multi-concern management as compared with the ProActive/GCM BS prototype. The ProActive/GCM prototype requires a more complex runtime and does not support multiple AMs in a single BS. LIBERO, like the ProActive/GCM BS implementation, uses the DROOLS [10] library middleware to implement autonomic managers' control cycles.

3.1 LIBERO Base Mechanisms

LIBERO implements component deployment on remote nodes using a small Java RMI-based runtime.This runtime allows deployment of LIBERO components and management of their life cycle. Management activities access the runtime to check

BS name	Features
Sequential	models sequential code, no actuator supported in AC, provides service time and executed task number through monitoring AC interface
Farm	models embarrassingly parallel stream parallel computations, constructor parameter used to pass the worker component class, AC supports increase/decrease parallelism degree actuators, provides service time, total task number and number of workers through the AC monitoring interface
Pipeline	models computations organized in stages, constructor parameters used to pass stage component classes, no actuator supported in AC, provides service time and total task number through monitoring AC interface

Fig. 2. LIBERO Behavioural Skeletons

machine dependent parameters peculiar to the node where the runtime is running, and may also access parameters associated with other nodes of the system, if needed.

The functional interfaces of LIBERO components are implemented using permanent Java TCP socket connections (either normal or SSL connections, depending on security requirements), with the use of serialisation for input/output object delivery between BS components. These permanent TCP connections imply the use of a discovery mechanism to locate the distributed components. Implementation of this mechanism assumes a global naming scheme for the components. A centralized multicast discovery component is used as a *Nameserver*.This component allows registration, removal and lookup requests using the specified component IDs. *Non-functional* interfaces (those related to BS managers) need stronger expressiveness and ease of use, and thus are implemented using RMI.

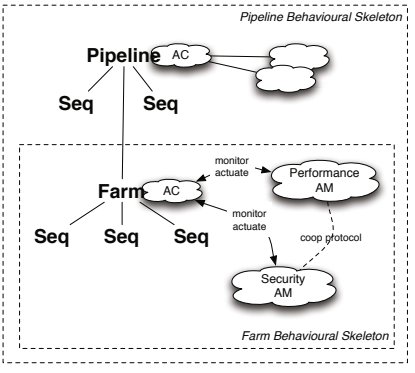
3.2 LIBERO BS

The LIBERO BS framework is provided as a set of classes [15]. A `BehaviouralSkeleton` class provides the common mechanisms (such as those needed for registration/removal of sub-components) and interfaces of BS and can be extended to implement new BS. Table 2 summarizes the main features provided by LIBERO.

Multiple managers, specialized by their contracts, can be declared using the appropriate LIBERO classes and associated with the same LIBERO BS. The actions of these cooperating AM are coordinated by means of the two-phase protocol proposed in [4]. The AM behaviour (that is, its contract) is expressed in terms of JBoss rules. The DROOLS rule pre-conditions are evaluated using the parameters monitored through the BS AC interface. Actions of the rules eventually fired by the DROOLS rule engine are executed using the BS AC interface.

The consensus protocol is implemented using JBoss rules and allows use of runtime values as contract parameters. As a consequence, the protocol is not embedded in the manager code but rather in the rule language. Fig. 3 (right) shows a sample JBoss rule. This is the rule fired when a new worker is added to a farm due to a breach of contract (fewer than 8 workers in the farm). The action part of the rule consists in setting up and broadcasting the consensus request.

Autonomic controllers provide mechanisms to monitor BS behaviour and to actuate manager decisions on the embedded skeleton. In particular, each AC implements the `executeOperation` and `getMeasure` methodsto change/export




```
rule "FarmPerformanceManagerRuleToAskForConsensus"
 when
  $farm: AutonomicControllerInterface()
  $manager: AutonomicManagerInterface()
  $sample: String() from
       $farm.getMeasure(Measures.NEXT_AVAILABLE_MACHINE)
  $sample_numworker: Integer() from
       $farm.getMeasure(Measures.TOTALWORKERS)

  not(exists(ContractParamValue(name ==
       MulticoncernBroadcastCodes.BCAST_REQUEST_WAIT_ACK)))
  not(exists(ContractParamValue(name ==
       MulticoncernBroadcastCodes.PREPARE_BCAST_COMMAND)))

  eval(((Integer) $sample_numworker) < 8)
then
  $manager.setContractParam(
       MulticoncernBcastCodes.PREPARE_BCAST_COMMAND, "");
  $manager.setContractParam(
       MulticoncernBcastCodes.BCAST_PARAM,
       CommandCode.INCREASE_PARALLELISM);
  $manager.setContractParam(
       MulticoncernBcastCodes.BCAST_SECOND_PARAM,
       $sample);
 end
```

Fig. 3. Sample use case application (left) and Sample JBoss rule (right)

the internal execution state. The AC also implements methods for accessing machine dependent parameters, fetched from the runtime support of the node.

Machine dependent properties are made accessible through the runtime support; these properties are described in an XML file parsed at startup by the runtime. The configuration file may host metadata relative to properties of the machines used for program execution.

4 Experimental Results

A set of experiments to assess LIBERO functionality and efficiency has been performed on an Intel/Linux cluster, with Java (version 1.5 or higher) and JBoss DROOLS (version 5.0). The nodes in the cluster were interconnected by FastEthernet and NFS was used. Here we report on one experiment illustrating multiple non-functional concern management in LIBERO.

This experiment uses a synthetic application structured as a three stage `Pipeline` component, as depicted in Fig. 3 (left): the first and the third stages are `Sequential` components, while the second stage is a `Farm` component. Each component is placed on a different node in the cluster and 3 machines, each running the LIBERO runtime are assigned as resources for the `Farm` workers.

The scenario under test is the following. *Both a performance contract and a security contract have been supplied by the user.* The performance contract requires that a given number of workers (8) be employed and that that level be maintained. It can be ensured by recruiting increasing numbers of resources to the point where the required number is operating. New resources may also be dynamically recruited for the computation in the event that existing ones become less effective due to temporary overloads or faults. The security contract demands that, where nodes are recruited from external, possibly unreliable domains, such nodes must be suitably secured by, for example, encrypting data

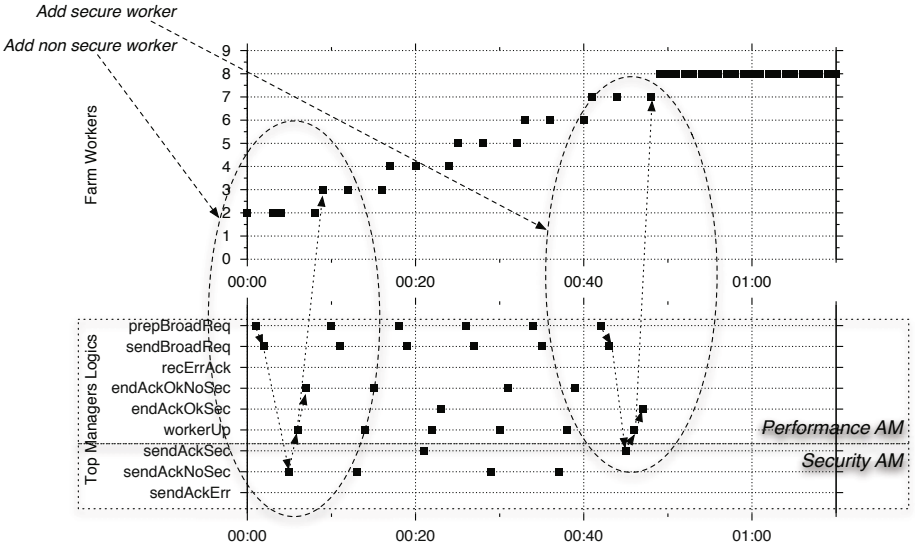


Fig. 4. Event distribution over time (secs from system startup). *W.r.t. Fig. 1:* prepBroadReq *corresponds to* ①, SendBroadReq *to* ②, SendAckSec/SendAckNoSec *to* ③/④, workerUp *to* ⑤/⑥ *and* endAckOkNoSec/endAckOkSec *to end of* ⑥.

and code communications; nodes internal to the user domain may be considered secure. Thus, if the performance manager identifies failure of the performance contract it will prompt the recruitment of further resources. If some of these are in an external domain the security manager may in turn demand the securing of communications with such potentially unsafe resources.

To implement this scenario, two autonomic managers are associated to the Farm component, one handling security and the other performance. The run time nodes host metadata classifying each node that may be recruited as secure or insecure.We used both secure and insecure nodes in the experiment to check both types of answers from the consensus phase: simple ACK (i.e. accept recruitment of a new node to host a `Farm` worker implemented using plain TCP sockets) and conditional ACK (i.e. accept recruitment of the node provided SSL sockets are used for communications).

The life cycle of the managers (the period used to run the DROOLS engine) is set to 500ms so that the plot of the runtime is sufficiently discrete to allow observation of the events, but smaller life cycles are possible down to 100ms.

In the use case we start the Farm component with two workers. The performance manager immediately detects a contract violation and asks the other managers for permission to add another worker. If other violations are encountered then the same set of operations is applied repeatedly, until no further violation is encountered.

The plot in Fig. 4 is automatically generated (but for arrows and ovals, added for clarification) from the application log files and shows evolution of the Farm component and the distribution of manager events over the same period of time.

As can be seen, consensus is sought and achieved according to the two phase protocol. In some cases workers are added using plain TCP connections (workers that happen to be placed on "secure" nodes – see on the right of Fig. 4). In other cases, the security manager detects that resources identified to host new workers are not secure and so it requests `property(Security)` in the ACK message. At this point the performance manager changes the plan used to add the worker from that employing plain TCP to one incorporating secure SSL connection, and eventually recruits the new worker using this modified action plan.

Overall, the consensus protocol takes an overhead of at most 4 manager life-cycles plus the execution time of the rules, which depends only on the communication overhead between managers. This gives a total overhead time of $T_{overhead} = 4 * (T_{LyfeCycle} + T_{Com})$, where T_{Com} is the average number of RMI calls * average RMI latency. In this simple case the entire reconfiguration of the system takes 45s, and reconfiguration time for worker allocation on average (including decision making and synchronization) is about 5 secs (including about 2 secs of idle time spent waiting 4 times for the next iteration of the control loop). These times are of the same order of magnitude as the times spent in the ProActive/GCM BS prototype to achieve an unmediated reconfiguration (i.e. a reconfiguration decided autonomically by a single, uncoordinated manager), which underlines the "lightweight" nature of LIBERO.

5 Related Work

The IBM blueprint paper on autonomic computing has already established, in a slightly different context, the need to orchestrate independent autonomic managers [12]. In [9] strategies to handle performance and power management issues by autonomic managers are discussed. However the approach is much more oriented to the generic combination of target functions relating to the two non-functional concerns considered, rather than to the constructive coordination of the actions planned by the two managers.

A framework that can be used to reason on multiple concerns was introduced in [13]. Based on the concepts of state and action (i.e. state transition) adopted from the field of artificial intelligence, this framework maps three types of agenthood concepts (action, goal, utility-function) into autonomic computing policies. Action policies may produce and consume resources, which are used by a *resource arbiter* (i.e. a super manager) to harmonize conflicting concerns. The framework does not, however, provide specific support for policy design and distributed management overlay.

A similar approach was followed in [5], which also exploits the same policies (action, goal, utility-function) defined on the *state* and *configuration* space of the system. These policies are extended with *resource-definition* policies, which specify how the autonomic manager exposes the system to its environment; this makes it possible to dynamically extend manager knowledge with other resources/parameters, possibly coming from other managers, thus supporting management overlay.

6 Conclusion

LIBERO supports the implementation of behavioural skeletons with multiple autonomic managers, each managing a different non-functional concern, and runs on any distributed architecture supporting Java. The prototype allows investigation of coordination aspects of autonomic management of non-functional concerns. The lightweight implementation of LIBERO, and in particular of the monitoring and actuator mechanisms implemented in the autonomic controllers, allows us to experiment with various consensus building strategies without being burdened by the complexities of fully-fledged distributed/parallel implementations, such as that provided by the ProActive/GCM BS implementation.

References

1. Aldinucci, M., Campa, S., Danelutto, M., Dazzi, P., Kilpatrick, P., Laforenza, D., Tonellotto, N.: Behavioural skeletons for component autonomic management on grids. In: CoreGRID Workshop on Grid Prog. Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments, Heraklion, Greece (June 2007)
2. Aldinucci, M., Campa, S., Danelutto, M., Vanneschi, M., Dazzi, P., Laforenza, D., Tonellotto, N., Kilpatrick, P.: Behavioural skeletons in GCM: autonomic management of grid components. In: Baz, D.E., Bourgeois, J., Spies, F. (eds.) Proc. of Intl. Euromicro PDP 2008: Parallel Distributed and Network-Based Processing, Toulouse, France, pp. 54–63. IEEE, Los Alamitos (February 2008)
3. Aldinucci, M., Danelutto, M., Kilpatrick, P.: Autonomic management of non-functional concerns in distributed and parallel application programming. In: Proc. of Intl. Parallel & Distributed Processing Symposium (IPDPS), Rome, Italy (2009)
4. Aldinucci, M., Danelutto, M., Kilpatrick, P.: Autonomic Management of Multiple Non-Functional Concerns in Behavioural Skeletons. In: Desprez, F., Getov, V., Priol, T., Yahyapour, R. (eds.) Proc. of the CoreGRID Symposium 2009, Delft, The Netherlands. CoreGRID, pp. 89–103. Springer, Heidelberg (August 2009), http://www.di.unipi.it/∼aldinuc/papers/2009_CGSymph_Autonomic_BeSke.pdf, ISBN: 978-1-4419-6793-0, doi:10.1007/978-1-4419-6794-7_8
5. Calinescu, R.: Resource-definition policies for autonomic computing. In: Proc. of the 5th Intl. Conference on Autonomic and Autonomous Systems (ICAS), pp. 111–116. IEEE, Los Alamitos (April 2009)
6. Cole, M.: Bringing skeletons out of the closet: A pragmatic manifesto for skeletal parallel programming. Parallel Computing 30(3), 389–406 (2004)
7. The CoreGRID home page (2007), http://www.coregrid.net
8. CoreGRID NoE deliverable series, Institute on Programming Model. Deliverable D.PM.04 – Basic Features of the Grid Component Model (February 2007) (assessed)
9. Das, R., Kephart, J.O., Lefurgy, C., Tesauro, G., Levine, D.W., Chan, H.: Autonomic multi-agent management of power and performance in data centers. In: Proc. of the 7th Intl. Conf. on Autonomous Agents and Multiagent Systems (2008)
10. Drools 5 - The Business Logic integration Platform (2010)
11. GridCOMP Project. Grid Programming with Components, An Advanced Component Platform for an Effective Invisible Grid (2008), http://gridcomp.ercim.org

12. IBM Corp. An Architectural Blueprint for Autonomic Computing (2005), http://www-01.ibm.com/software/tivoli/autonomic/
13. Kephart, J.O., Walsh, W.E.: An artificial intelligence perspective on autonomic computing policies. In: Proc. of the 5th Intl. Workshop on Policies for Distributed Systems and Networks (POLICY 2004). IEEE, Los Alamitos (2004)
14. ProActive home page (2009), http://www-sop.inria.fr/oasis/proactive/
15. Xhagjika, V.: Implementation of a prototype for experimenting with autonomic hierarchical managers in java (thesis, in italian). Dept. of Computer Science, Univ. of Pisa, Italy (December 2009)