MiniMASC: A Framework for Diverse Autonomic Adaptations of Web Service Compositions

Qinghua Lu\textsuperscript{1,2}, Vladimir Tosic\textsuperscript{2,1}
\textsuperscript{1}School of Computer Science & Engineering, Uni. Of New South Wales, Sydney, Australia
\textsuperscript{2}Management Complexity Group, NICTA, Sydney, Australia
Qinghua.Lu@nicta.com.au, vladat@computer.org

Abstract—When various technical and business changes related to Web service compositions occur, there is often a need for runtime adaptation with minimal human intervention. While many research projects work on particular types of such adaptation, much more research is needed on decision making for diverse autonomic adaptations, particularly to maximize business (as opposed to technical) metrics. Our MiniMASC middleware is a framework for diverse autonomic adaptations of Web service compositions, focusing on supporting such advanced decision-making algorithms. MiniMASC is relatively simple, light weight, modular, and extensible. It uses the WS-Policy4MASC policy language that can describe all information necessary for different types of adaptation. After presenting our classification of different types of decision making in adaptation of Web service compositions, this paper discusses how MiniMASC (with WS-Policy4MASC) can be used for implementing algorithms maximizing business metrics. Our tests show that the implementation of MiniMASC has a satisfactory performance and scales well.

Keywords-autonomic computing, dynamic adaptation, decision making, Web service management, business-driven IT management

I. INTRODUCTION AND MOTIVATION

Service-oriented architecture (SOA) \cite{1} implemented with Web services is currently the most popular approach to build information technology (IT) systems for complex business processes. In long-running business processes, changes (both business and technical) happen relatively often and require adaptation at run-time, with minimal human intervention. This adaptation usually can be done in several ways and advanced decision making is needed to determine how to proceed with the adaptation actions. However, it is becoming impossible for human managers to manage modern large-scale IT systems and services (such as Web service compositions implementing complex, long-running business processes) because they cannot understand multifaceted interdependencies between a huge number of diverse components of such systems and services. There is a need that software performs most of the management activities automatically, with only minimal human supervision. Autonomic computing \cite{2} is an approach towards solving complexity in IT system/service management, where IT systems self-manage themselves according to high-level policies.

The traditional IT system and service management approaches mainly focus on functional correctness and optimization of technical metrics, such as response time and availability. However, business metrics, such as profit and market share, are more important to business users of IT systems than technical metrics. For example, if the response time of a service changes from 10ms to 30ms, a business user is usually not very interested in the precise new response time, but wants to know the business impact of this change, e.g. how much it will cost her/his business. Business-driven IT management (BDIM) \cite{3} is a relatively novel research area that studies models, algorithms, and architectures for mapping business and technical IT metrics and making IT system/service management decisions best from the business viewpoint. Examples of business metrics are profit and market share, while examples of technical metric are response time and availability. Decisions in BDIM are made based on such mappings and are performed to maximize business value for the enterprise. Autonomic BDIM is a combination of autonomic computing and BDIM, which adds processing of business metrics to autonomic decision making components of IT management systems.

While there has been a lot of research in the area of adaptation of Web service compositions, there has been relatively little research on advanced support for autonomic business-driven decision making for such adaptation. There is a need for further studies of and experiments with various algorithms for autonomic business-driven decision making in adaptation of Web service compositions. In this paper, we present MiniMASC middleware, our framework for diverse autonomic adaptations of Web service compositions. MiniMASC is relatively simple, light weight, modular, and extensible. A particular strength of MiniMASC and a focus of this paper is support for exploration and exploitation of advanced autonomic business-driven decision-making algorithms.

This paper is organized in the following way. This section introduces the background and motivation of the paper. Section 2 classifies different types of decision making in adaptation of Web service compositions and analyzes to which extent the past works addressed these types. Section 3 gives an overview of the architecture of MiniMASC and how it uses WS-Policy4MASC. Section 4 presents the autonomic BDIM support in MiniMASC. Section 5 discusses our evaluation of MiniMASC. Section 6 concludes the paper.
II. CLASSIFICATIONS OF DECISION MAKING FOR ADAPTATION OF WEB SERVICE COMPOSITIONS AND POSITIONING OF PAST RELATED WORK

There is often a need to adapt Web service compositions (and business processes they implement) to changes, while the system executes at run-time. There are many possible needs and ways for such adaptation. Many projects work on adaptation of Web service compositions. However, they mainly work on some types of adaptation. Decision making algorithms that can be used in deciding which adaptation actions to execute (often, selected among several possible alternatives) can differ both across different types of adaptation and within a particular type of adaptation. [4] concluded that further research is needed on development of powerful decision making algorithms for diverse autonomic adaptations, particularly to optimize business metrics. To be able to compare and combine different algorithms, it is necessary to examine and classify various characteristics of the involved decision making. Such classification is also useful to guide development of new algorithms and corresponding framework for autonomic adaptation of Web service compositions. Thus, we developed a novel classification of decision making for adaptation of Web service compositions.

Our classification was inspired by the 6 questions that are widely used in journalism (and some other areas) as a guide for completely describing a situation: “Who?”, “What?”, “When?”, “Where?”, “Why?”, and “How?”. Collectively, these 6 questions are known as “5WH questions”. We examined the meaning of these 6 questions in the context of adaptation for Web service compositions and determined the following 6 dimensions of classification of types of decision making in adaptation of Web service compositions: causes of the need for adaptation (answering the question “When?”), goals of the adaptation (“Why?”), scope of the adaptation (“Where?”), adaptation decision making actors (“Who?”), planning of the adaptations (“How?”), and changes caused by the adaptation (“What?”). We summarized this in Table 1.

Note that these 6 classification dimensions are general enough to be applicable to other adaptations of IT systems. However, while some types of adaptation that we identified within these categories can be also observed in adaptation of different IT systems, some are specific for adaptation of Web service compositions (and business processes they implement). In the following paragraphs, we will present the main types that we identified in our classification, for each of the 6 dimensions.

<table>
<thead>
<tr>
<th>“5WH” Question</th>
<th>Classification Dimension for Decision Making in Adaptation of Web Service Compositions</th>
</tr>
</thead>
<tbody>
<tr>
<td>When?</td>
<td>Causes of the need for adaptation</td>
</tr>
<tr>
<td>Why?</td>
<td>Goals of the adaptation</td>
</tr>
<tr>
<td>Where?</td>
<td>Scope of the adaptation</td>
</tr>
<tr>
<td>Who?</td>
<td>Adaptation decision making actors</td>
</tr>
<tr>
<td>How?</td>
<td>Planning of the adaptations</td>
</tr>
<tr>
<td>What?</td>
<td>Changes caused by the adaptation</td>
</tr>
</tbody>
</table>

We will also point out which types are covered by representative past works. Since there are many research projects on particular types of adaptation of Web service compositions and business processes they implement, our goal here is not to survey all past publications, but only to position some representative related works.

Causes of the Need for Adaptation. Adaptation of Web service compositions can be caused by a number of reasons, which can be classified into:

- Technical reasons [5-12], which include: some services do not run (e.g., their implementations cannot execute), some services run with errors (e.g., due to an unexpected input), some service cannot be accessed (e.g., because they are overloaded), performance of some services is not optimal (e.g., their availability drops), changes in connecting infrastructure (e.g., there are network changes), some services have been replaced (e.g., with incompatible new services), changes in non-business context (e.g., new technology emerged).

- Business reasons [7, 9], which include: business goal changes (e.g., “double the current profit goal for this year”), business strategy changes (e.g., “change focus from gold consumers to silver consumers”), business structure changes (e.g., business partner changes), business rule changes (e.g., “triple number of server resources for gold consumers”), business customer changes (e.g., customer preferences have changed), business context changes (e.g., new competitors emerged).

The vast majority of past works researching adaptation of Web service compositions primarily focused on adaptations triggered by technical reasons. On the contrary, only a few research projects examined the impact of business causes, but did not classify them in detail. It is important to understand various causes of the need for adaptation, because they often require different monitoring, decision making, and adaptation execution. For example, if a management system does not monitor business-related events, it will not be able to recognize the adaptation needs they cause, so it will not be able to react and adapt appropriately.

Goals of the Adaptation. We will illustrate how different goals placed on adaptation of Web service compositions influence decision making on an example when there the Web service composition uses Web service X that has contractual obligation to provide response time under 10ms at the cost of AU$10 per month. Assume that Web service X is no longer available, but there are 4 possible replacement Web services (none of which is exact replacement): Web service A promises to provide response time under 10ms at the cost of AU$12, Web service B promises response time under 10.5ms at the cost of AU$10, Web service C promises response time under 7ms at the cost of AU$13, and Web service D promises response time under 12ms at the cost of AU$4. Assume further that the penalty for not meeting the response time is always AU$0.10 per incident and that there are no more than 200 such invocations per month. We classified the goals of adaptation into:
• Stick to the contract [13-14]. In this case, the goal is to reinstate the situation before the change, particularly the technical metrics in the original contract (e.g., a service level agreement – SLA), as much as possible. Then, the Web service A is the best replacement, irrespective of the higher cost than original AU$10.

• Minimize the impact of the change that caused the adaptation [15]. This goal differs from the above-mentioned in that the impact of the adaptation is considered somewhat more broadly and could consider prices and penalties. In the studied example, it is likely that Web service B would have minimal impact (its price is lower than for the Web services A and C).

• Achieve the maximal technical metrics (quality of service – QoS) [6, 16-17]. The goal here is to maximize technical metrics, but possibly with some cap on costs. In the studied example, Web service C provides the best response time.

• Achieve maximal business value [18]. This goal here is to make a decision that is best from the business viewpoint (as opposed from the technical viewpoint). Business value goes beyond simple consideration of prices and penalties to be paid and can consider additional business metrics, such as market share and customer satisfaction, because of their impact on future profits. In the studied example, it is possible that the Web service D would turn out the best from the business viewpoint, because it is considerably cheaper than the alternatives (while its guaranteed response time is somewhat higher, it is not too high).

The past adaptation decision making algorithms predominantly choose adaptation that maximizes technical metrics. In order to better fulfill business needs, it is advantageous to focus on maximization of business value – this is an area where there are still many open research issues [4].

Scope of the Adaptation. Broadly speaking, we identified two different scopes of adaptation for Web service compositions:

• Schema level adaptation. Here, the business process (i.e., Web service composition) description (e.g., in the Web Services Business Process Execution Language – BPEL), is modified. All instances of the Web service composition will be affected. Schema level adaptation includes two different sub-types: one same adaptation for all consumers [8-9, 17], or different adaptations for different classes of consumer [13].

• Instance level adaptation. Here, different running instances can be adapted in different ways (or might not be adapted at all), even if they have similar characteristics such as class of consumer and current position/state within the running process. The different adaptations for similar instances are done to maximize business value with existing resources. For example, if there are 10 instances with the similar characteristics and there is not enough resources to provide 99.99% availability to all of them, then some (e.g., 6 – this depends on the available resources) are adapted to get this high availability, while the rest (in this example, 4) are adapted to get lower availability (e.g., only 99%).

Most previous research projects focus on the one same adaptation for all consumers. Only a few research projects focus on different adaptations for different classes of consumer, while instance level adaptation is studied rarely.

Adaptation Decision Making Actors. Adaptation decisions can be made manually, semi-automatically, or automatically. Correspondingly, the adaptation decision making actors can be classified as: human administrators, human administrators with significant help from computer systems [19], computer systems with minimal help from humans [6, 11, 19].

While in industry practice adaptation decision making is still mainly done by human administrators, the vast majority of the research (including our own) focuses on the automatic and, to a lesser degree, semi-automatic adaptation. While automatic adaptation is the vision of autonomic computing, it is very hard to achieve. Thus, it is likely that in the near future the industry will mostly use semi-automatic adaptation.

Planning of the Adaptations. Adaptation decision making can be also classified according whether only a single action or a more complex plan is to be determined:

• Single action [16, 20-21]. Here, one best adaptation action (possibly with several sub-actions, but with clear ordering between them) is chosen from several possible alternatives.

• Single plan [22-23]. Here, a more complex plan (e.g., selection of a group of more loosely coupled sequential actions, possibly with some flow control) is made.

• Multiple parallel plans. This is relevant when different classes of consumer or even different instances are to be adapted in a different way (see discussion for the “Scope of the Adaptation” above). In such cases, there is a need for separate plans (some of which could be single actions) for these different adaptation and for coordination (e.g., synchronization) of these plans.

The majority of previous research projects worked mainly on deciding single actions for adaption of Web service compositions. Due to the complexity of determining longer-term plans, there is relatively fewer works on plan generation, particularly on creation and coordination of multiple parallel plans.

Changes Caused by the Adaptation. Different aspects of system configuration and execution could be affected by adaptation of Web service compositions:

• Changing the structure of the implemented business process [6, 17, 21, 24], which can be creating a new business process or modifying the current business process, including changing the Web service composition (e.g., adding new services, removing some services) and changing individual services (e.g., replacing a service, changing implementation of a service, reconfiguring some parameters used by a service).
• Changing execution of the business process [5, 8-9, 11], by performing actions such as retry, skip, rollback, and termination of the business process instance.

• Changing contracts with customers [25-26]: switching customers between existing contracts, making new contracts (e.g., making new functional contracts or SLAs). Changing existing functional contracts (e.g., adding new functionality), changing existing extra-functional contracts (e.g., response time limits and prices in an SLA).

Different types of changes are well supported by the past research literature, although very rarely a comprehensive approach to supporting all these types of changes is adopted. This identification of various changes can be used as a guide for outputs of decision making algorithms in autonomic systems.

III. ARCHITECTURE OF MINIMASC

MiniMASC is a relatively simple, light weight, modular and extensible middleware supporting different types of adaptation of Web service compositions (and, potentially, other IT systems). It can support all types of adaptation that we have identified, although not all of them are fully supported in the current prototype (the emphasis in the current prototype is on supporting autonomic business-driven IT management to maximize business value, as we will discuss in the next section). Fig. 1 shows the architecture of our autonomic management system for Web service compositions, in which MiniMASC is the most important component. There are 3 groups of components and artifacts, shown in Fig. 1, with different background colors.

On the right-hand side (without any background color) are components that our autonomic system integrates with, without re-implementing them: the executing (running) managed system, third-party modules for monitoring the managed system, and third-party modules for executing chosen adaptation actions. The managed system can be a Web service or Web service composition (business process) system, but it can also be a cloud computing, grid computing, or cluster computing system. MiniMASC does not implement these modules (but integrates with them) because there are many research frameworks (such as MASC [25], based on the experiences with which we designed MiniMASC) and commercial products that address these issues relatively satisfactorily (although certainly not perfectly). Being a research group with limited human resources, we decided to focus our research on how to support diverse decision making in adaptation of Web service compositions and to integrate hooks for any external (third-party monitoring and adaptation execution modules).

In the centre of Fig. 1 (with the most prominent background) are the main parts of our autonomic management system of Web service compositions. The information guiding management decisions is specified as policies in WS-Policy4MASC [4], our extension of the Web Services Policy Framework (WS-Policy) industrial standard. WS-Policy4MASC [4, 27] is a very powerful and extensible policy language that can describe various adaptations and all information necessary for adaptation decision making. The information guiding adaptation decisions is specified as policies (which can be also viewed as business rules) in WS-Policy4MASC. In WS-Policy, a policy is defined as a collection of policy alternatives, each of which is a collection of policy assertions. WS-PolicyAttachment defines how to associate a policy with subjects (e.g., Web services) to which it applies. WS-Policy4MASC adds 5 new types of WS-Policy policy assertions: goal (prescribing conditions to be met, e.g., desired response time), action (listing groups of adaptation actions to be performed), utility (containing business value metrics (BVMs) for particular situations), probability (specifying probabilities of occurrence), and meta-policy assertions (describing which values are important for adaptation decisions). In addition to these 5 new types of policy assertions, WS-Policy4MASC has a number of auxiliary constructs, specifying ontological meaning, monitored QoS metrics, monitored context properties, states, state transitions, events, schedules, applicability scopes, and various types of expression. The most important of these auxiliary constructs is “When”. It specifies 1 or more states in which a policy assertion is triggered, 1 or more events (e.g., Web service operation executed) that can (mutually independently) trigger the policy assertion, and an optional filtering Boolean condition to be satisfied. As explained in [4, 27], WS-Policy4MASC enables specification of detailed information necessary for run-time policy-driven management and overcoming some limitations of WS-Policy.

Based on monitored data about execution of the managed Web service compositions and according to policies in WS-Policy4MASC, MiniMASC decides which adaptation approach is best from the different viewpoints (e.g. technical viewpoint, business viewpoint). One of the main goals for development of MiniMASC, was to make it relatively simple, light weight, modular and extensible. In this way it differs from its predecessor, the MASC middleware [25], which aimed at comprehensiveness of support for Web service composition management activities (e.g., it implemented powerful modules for monitoring and adaptation execution) which made it complicated and harder to extend. Our MiniMASC was implemented in Java and PostgreSQL (contrary to MASC, which was implemented in C# and Microsoft .NET 3.5).

In MiniMASC, WS-Policy4MASC policies are stored in the Policy Repository, which is a Structured Query Language (SQL) database. The module Policy Parsing takes as input WS-Policy4MASC and WS-PolicyAttachment policy files and generates corresponding INSERT statements in SQL. MiniMASC supports the latest WS-Policy4MASC version 0.92 [4], significantly more powerful than the WS-Policy4MASC version 0.8 used in MASC. Another SQL database in MiniMASC is Database of Monitored Data, which stores run-time data about monitored technical metrics (e.g., measured response time, calculated availability), business metrics (e.g., paid prices and penalties), and events that come from the third-party monitoring modules. The module Processing of Monitored Data converts the received data into corresponding INSERT statements. Based on the recent monitored data (particularly events) and historical information
stored in the Database of Monitored Data, the module Determining Triggered Policies determines which WS-Policy4MASC policy assertions should be executed. We are currently integrating the Policy Learning module that enables run-time adjustment of policies stored in the Policy Repository, based on historical information in the Database of Monitored Data. Due to the complex nature of WS-Policy4MASC policies, this module will not learn completely new policies, but only adjust numerical values, e.g., response time limits, if they are judged to be unrealistic (e.g., response time limits are constantly exceeded, after attempting different adaptations).

The most important part of MiniMASC is the Policy Conflict Resolution module, which implements various adaptation decision algorithms (discussed in the next section) deciding whether the triggered WS-Policy4MASC action policy assertions (APAs) conflict (i.e., represent mutually exclusive adaptation approaches) and, if they do, which of them should be executed. There are six sub-modules in this module: Meta Policy Processing, Constraint Checking, Utility Calculations, Ordering of Alternatives, Tiebreaking, and Policy Determination. The Meta Policy Processing sub-module processes the relevant rules specified in the WS-Policy4MASC meta-policy assertions, which describe rules how to determine the “best” among possible adaptation approaches. The constraint checking sub-module checks whether the triggered APAs satisfy all given constraints (e.g., cost limits). The Utility Calculations sub-module calculates utility (e.g., the total business value metric – BVM) for each the triggered APAs that satisfy all constraints, according to the rules specified in the WS-Policy4MASC meta-policy assertions. The Ordering of Alternatives sub-module orders all triggered APAs that satisfy all constraints into a descending order of their utility, to determine which of these APAs provides the highest utility (e.g., maximal technical performance, maximal business value). The Tiebreaking sub-module is used when utility of 2 or more conflicting APAs are close enough so that difference is considered negligible – in this case, additional information about the consequences of these APAs is considered to make the most appropriate choice. The Priority Determination sub-module makes the final selection of the APA to be executed.

On the left-hand side of Fig. 1 (with light gray background) are optional components and artifacts in our system. Their main purpose is to help users in creation of WS-Policy4MASC (and WS-PolicyAttachment) files, which can be very complex. Without them, WS-Policy4MASC files are written by system administrators or software engineers, in consultation with business analysts (providing necessary business-related data). However, our research group previously developed software [28] that semi-automatically creates WS-Policy4MASC files from UML models annotated with original stereotypes compatible with WS-Policy4MASC policies. The same work also enabled annotation of monitored data on these extended UML models, to facilitate decisions about design-time evolution (e.g., re-design) of the managed systems.

IV. AUTONOMIC BDIM SUPPORT IN MINIMASC

[29] concluded that autonomic BDIM is a promising research area, with many unsolved challenges. However, it is
still an underexplored area, particularly in applications to management of Web services and their compositions. As mentioned above, maximization of business value (as opposed to maximization of technical metrics) is important for business users and autonomic execution of adaptations is needed to minimize involvement of human administrators (who might not be available, might take too long to analyze complex situations and respond, or are too expensive). Therefore, we choose support for autonomic BDIM as the focus of our research, with detailed support in the current MiniMASC (and WS-Policy4MASC) design and prototype implementation.

Fig. 2 shows the basic BDIM conflict resolution algorithm that we implemented in the Policy Conflict Resolution module of MiniMASC. The algorithm first loops through all conflicting APAs (left part of Fig. 2) to check whether each of them satisfies all given constraints and, if yes, to calculate sum of all business value metrics (BVMs) for this APAs. The APAs that satisfy all constraints are added to the list, along their summary BVM. If none of the conflicting APAs satisfies the constraints, the resulting list will be empty and an exception is thrown (e.g., human administrators are notified to resolve the issue). The APAs in the list are ordered based in the decreasing value of their summary BVM. If tiebreaking is specified the tiebreaking information is calculated and applied in cases when the difference between summary BVMs is too small to be of importance. There could be several rounds of tiebreaking, each resulting in reordering of the list of APAs. After all relevant tiebreaking (if any) is applied, first APA in the list will have the highest summary BVM while satisfying all constraints and tiebreaking rules. This APA is returned as the adaptation action (which could contain several sub-actions) to be executed.

It can be noticed in the algorithm from Fig. 2 that there are several modules that calculate various information. There are commonalities between these calculations, because all of them have to sum some BVMs. To increase modularity and reusability, we designed and implemented a general module BVMs Summation and a number of modules that specialize it (through inheritance). We will mention 4 of such specializations. Checking of cost limits is a common constraint, so the appropriate algorithm is implemented in Cost BVMs Summation. Summation of all BVMs can be done in different ways (e.g., without or with probabilities of occurrence) and we experimented with several of them, resulting in the specialization modules All BVMs Summation with Probabilities and All BVMs Summation with Longer-Term Strategic Information. Application of tiebreaking rules also requires specific summation of BVMs, implemented in the module Tiebreaking BVMs Summation. This is only 1 example how in design of MiniMASC we tried to increase modularity, reusability, and extensibility.

The algorithm in the module All BVMs Summation with Probabilities is shown in Fig. 3 in detail. Business value metrics (BVMs) for particular situations are specified within WS-Policy4MASC utility policy assertions (UPAs). A number of UPAs can correspond to consequences of executing a particular APA. Thus, the first step of the algorithm in Fig. 4 matches all UPAs in the Policy Repository to find those that are relevant for the currently examined APA (one of the conflicting APAs from the algorithm in Fig. 2). The BVMs in WS-Policy4MASC UPAs can be given in the form of arithmetic-with-currency-unit expressions (possibility with variables), so the next step of the algorithm calculates (for all BVMs in all relevant UPAs) the monetary amounts resulting from these expressions and converts them into a common currency (using current data from an external currency conversion service). Then, the algorithm checks whether use of probabilities is enabled. If it is, each of the BVMs (in all relevant UPAs) is multiplied with 2 probabilities – the likelihood that preconditions of the current UPA will be met and the likelihood that the given values are correct estimates (when the latter is not 100%, then there must be several compatible UPAs whose probabilities add to 100%). At the end, the algorithm determines the summary result that is returned to the caller. However, this is not a simple sum, but a
Figure 3. The algorithm in the module All BVMs Summation with Probabilities.

data structure with 8 values, each summing BVMs of the same category. This is because in different circumstances (e.g., business strategies), different business value metric categories are relevant for autonomic BDIM decision making. This was explained in detail in [4, 25]. Due to the space limits, we will only note here that the 8 categories are: 1) financial agreed benefits, 2) financial agreed costs, 3) financial non-agreed benefits, 4) financial non-agreed costs, 5) non-financial agreed benefits, 6) non-financial agreed costs, 7) non-financial non-agreed benefits, and 8) non-financial non-agreed costs.

The conflict resolution algorithm requires frequent use of the data, which would lead to constant calls to the database in order to retrieve the data. Therefore, the algorithm uses a large number of Java classes to store the data from the database. The data stored in Java classes are initialized before the algorithm is running. This not only improves the efficiency of the code (as calls to the database are relatively inefficient), but makes the code cleaner and easier to read and modify in future work.

V. EVALUATION

We evaluated MiniMASC in 3 aspects. First, we evaluated feasibility of the MiniMASC architecture and the developed original algorithms for autonomic BDIM through implementation of several proof-of-concept prototypes. Apart from the MiniMASC prototypes in Java, we re-implemented some modules in C# to check that feasibility does not depend on Java features. We found no problems with feasibility.

Second, we evaluated modifiability and extensibility of the MiniMASC architecture by designing and implementing different prototypes for different versions of the autonomic algorithms and observing how easy it is to perform such changes. In addition to the smaller variations of the basic MiniMASC architecture and algorithms presented in this paper, we have designed, partially implemented, and experimented with several major modifications of the MiniMASC architecture. One of them integrates MiniMASC modules with the Application Server Framework (ASF), elaborating ideas presented in [30]. (Problems in using the original MASC modules for this integration were one of the motivations for development of MiniMASC.) Another builds into the MiniMASC modules (particularly, the decision making algorithms) storage and processing of longer-term strategic information specified in the Business Motivation Model (BMM), along the ideas from [31]. In both cases, changes to the MiniMASC functionality were significant, but the MiniMASC modularity with clean interfaces helped to localize modifications and extensions. Notably, the policy conflict resolution module is composed of sub-modules with different functionality. It is flexible and easy to add, remove, or modify these sub-modules to solve diverse decision making problems in adaptations of Web service compositions. Also, the BVMs Summations sub-module can be easily inherited and specialized for various specific calculations of BVMs. We also got feedback from undergraduate students that MiniMASC is much easier to understand and modify/extend than the original MASC middleware.

Third, we checked that MiniMASC is lightweight enough, in the sense that its performance and scalability (when the complexity of the decision-making problem increases) is satisfactory. In this respect, we conducted several performance and scalability tests, particularly focusing on the autonomic BDIM policy conflict resolution algorithm described in the previous section. For these tests, we used a Hewlett-Packard laptop model HP EliteBook 6930p with Intel Core 2 Duo CPU T900 2.53GHz processor and 4.00 GB of RAM memory, running 32-bit Windows Vista operating system. The results of a representative test are shown in Table 2. In this test, we increased the number of BVMs related to each of the 2 conflicting APAs. We started with 2 BVM, then increased to 32, and then to 64 (tests that we did for some other numbers of BVMs are not shown in Table 2). Using the Unix “time” utility, we measured the execution time of decision making in MiniMASC. Additionally, using the Java “System.currentTimeMillis()” call, we measure the execution time of the central policy conflict resolution algorithm. We repeated a large number of tests at different times of days and averaged their results. The most important information is that the execution time of the conflict resolution algorithm only slightly increased with the increased complexity of the test cases. This indicates to us that the algorithm was implemented efficiently and scales well. However, the overall execution time of decision making in MiniMASC showed higher increases – around 23% from test 1 to test 2, and around 19% from test 2 to test 3. We believe that this is primarily due to the increased processing of WS-Policy4MASC policies. We also did tests (not shown in Figure 2) with a greater number of conflicting APAs, but the impact of this complexity increase was also not high. It is important to note that in realistic application scenarios of MiniMASC the number of conflicting APAs will be low (typically 2, maybe a few more), while the number of relevant BVMs per conflicting APA will most likely will be much lower than 64. Therefore, we did not perform detailed additional tests with much higher number of
APAs and BVMs. In our opinion, our performance and test results for MiniMASC produced satisfactory results.

### TABLE II. MEASUREMENT RESULTS WITH INCREASING NUMBER OF BVMs RELEVANT FOR A CONFLICTING APA.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Execution Time of Decision Making in MiniMASC</th>
<th>Execution Time of the Conflict Resolution Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 BVMs</td>
<td>Average: 265 ms</td>
<td>Average: 15 ms</td>
</tr>
<tr>
<td></td>
<td>Range: 265-297 ms</td>
<td>Range: 15-16 ms</td>
</tr>
<tr>
<td>32 BVMs</td>
<td>Average: 327 ms</td>
<td>Average: 16 ms</td>
</tr>
<tr>
<td></td>
<td>Range: 327-343 ms</td>
<td>Range: 15-16 ms</td>
</tr>
<tr>
<td>64 BVMs</td>
<td>Average: 390 ms</td>
<td>Average: 16 ms</td>
</tr>
<tr>
<td></td>
<td>Range: 327-406 ms</td>
<td>Range: 15-16 ms</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS AND FUTURE WORK

Autonomic BDIM is an important and promising, but still underexplored area of research. We believe that development and exploration of various decision-making algorithms is the most important open sub-area of autonomic BDIM. In order to better understand various types of adaptations and corresponding decisions to be made by the autonomic BDIM software, we performed the classification presented in this paper. This classification also helped us to uncover various requirements for software performing diverse types of decision making for adaptation of Web service compositions and to identify several aspects (particularly autonomic BDIM related) that were not explored sufficiently in the past literature.

Based on the knowledge from this classification, survey of the related literature, and our previous experiences with the MAS middleware, we designed, implemented, and tested MiniMASC as a relatively simple, light weight, modular and extensible middleware framework for diverse autonomic adaptations of Web service compositions. It is important to note again that although the focus of this paper was on the autonomic BDIM support, MiniMASC can be used also for traditional decision making that maximizes technical metrics. This is primarily because WS-Policy4MASC can describe various adaptations and all information necessary for decision making of all types of adaptations for Web service compositions. Our evaluation using several prototype implementations showed that the proposed MiniMASC architecture and autonomic BDIM decision-making algorithms are feasible and easy to modify/extend. Furthermore, our performance and scalability tests showed that MiniMASC does not introduce unforeseen performance or scalability problems. We use MiniMASC in two ways. First, we use it to experiment with various new autonomic BDIM algorithms. Second, we implement integrations of MiniMASC with third-party monitoring and adaptation execution modules to enable use exploitation of our solutions in realistic settings. In this way, MiniMASC is not only a research middleware, but also potential deployable component in a (commercial) Web service composition management system.

Our ongoing work focuses on modifications and extensions of MiniMASC, e.g., the integration with the Application Server Framework and the extension processing longer-term strategic information specified using the Business Motivation Model (BMM). We prepare publications about these results. We also work on several other applications of MiniMASC, hopefully in a commercial setting (to achieve this, we actively look for industrial partners and already held some promising discussions).

REFERENCES


