Comparative Evaluation of the State-of-the-art on Approaches to Software Adaptation

Nor Hazilawati Awang
HeiTech Padu Berhad
Persiaran Kewajipan, Subang Jaya
Selangor
03-86013424
hazila@heitech.com.my

W M N Wan Kadir¹ and Shamsul Shahibuddin²
Software Engineering Dept.
Faculty of Computer Science and Information System,
Universiti Teknologi Malaysia
07-5532348
¹wnasir@utm.my, ²shamsul@utm.my

ABSTRACT
Software needs to evolve to ensure it continuously relevant in supporting the needs of an organization. Thus, software evolution is unavoidable for its survival. Due to rapid changes in business environment and advancement of technology, simplifying software evolution becomes more challenging and may involve high cost. Approaches in simplifying software evolution via software adaptation have been the subject of many current researches. Based on exhaustive literature review, we define four classifications of approaches to software adaptation. The four approaches are Architecture-based, Component-based, Middleware-based and Agent-based. In this paper we present the results of a systematic comparison on the state-of-the-art in software adaptation approaches mentioned earlier. Five evaluation criteria are defined to compare the said approaches. The evaluation criteria used are scalability, context-awareness, heterogeneity, performance and dynamic evolvability. The result of the evaluation is used to determine the best current approach to developing adaptive software in order to simplify software evolution. The evaluation result is also used as input for the development of a framework to simplify software evolution. High-level view of the framework is presented towards the end of the paper.

Categories and Subject Descriptors
D.2.11 [SOFTWARE ENGINEERING]: Software Architectures
D.2.13 [SOFTWARE ENGINEERING]: Reusable Software – Reuse models.

General Terms
Design

Keywords
Software evolution, software adaptation, architecture-based, component-based, middleware-based, agent-based.

1. INTRODUCTION
Software evolution phenomenon has the potential to present huge problems to an organization since it involves changes to software. For a long term survival of a software, evolution is inevitable [1]. With technology advancement and dynamic business environment, software evolution becomes more complex and difficult to manage. If not carefully planned, software evolution will occur with high cost. Due to this, effective approach to software evolution fired much interest in research community as organizations have a growing dependency on software. Managing software evolution via software adaptation has been the subject of many researches such as in [2-7]. Software evolution and software adaptation are connected processes. Oreizy et al describe extensively on the inter-connectedness between evolution management and adaptation management [8].

Research work in software adaptation ranges from the development of generic architectural framework to specific middleware for specialized domains. Many approaches to software adaptation have been proposed in recent years. In this paper, we classify four prominent approaches to software adaptation, namely, Architecture-based, Component-based, Middleware-based and Agent-based. In Section 2, we will briefly describe each approach. Explanation on the criteria used in the evaluation can be found in Section 3. In Section 4, the discussions on the evaluation result are presented. Conclusion and future research work are presented in Section 5.

2. APPROACHES TO SOFTWARE ADAPTATION
One of the main concerns in software adaptation is on the development of adaptive software. Adaptive software can be defined as software that is able to change its behavior according to the changes in its context [4]. In general, two main approaches have been identified by researchers for the implementation of adaptive software, namely compositional adaptation and parameterized adaptation. Compositional adaptation refers to the ability of the software to dynamically reconfigure itself at runtime to suit the changing operating environment [9]. Parameterized adaptation, or static adaptation, involves modification of program variables that directly affect the system behavior [9]. In this approach, adaptation is enabled during compile and start-up time [10]. Both approaches are useful for software adaptation. Four prominent approaches to software adaptation can be used to achieve compositional and
parameterized adaptation are identified. The approaches are Architecture-based, Component-based, Middleware-based and Agent-based. The following sections briefly described each approach.

2.1 Architecture-based Approach (Arc-based)
Architecture-based approach makes use of architecture model to achieve adaptation at run time. The architecture model specifies component composition and their interconnectivity [11]. Using this approach, developer’s view is abstracted out to focus on the entire systems instead of line-of-codes. This enables an integrated view of the whole systems. There are a number of ways in developing adaptive software using architecture-based approach including the use of Architecture Definition Language (ADL) [12, 13]. Another approach in architecture-based is Aspect-Oriented Modeling (AOM) [14].

Architecture-based approach to adaptation offers a number of benefits since it provide a global view of the system to be constructed. This enables system-level properties and constrain to the architecture to be exposed at an early stage thus adaptation aspect of the software can be planned earlier [15].

There are also some challenges in adopting this approach for the development of an adaptive system. Software architecture is typically used at the design time and it does not specify how the system is to be constructed. There is a gap between the architecture modeling and the actual construction of the adaptive software, therefore this gap must be carefully addressed. The gap can cause loss of knowledge on the software architecture since some of the information about system properties and constrain are not made explicit [7]. As a result, proper construction of the software becomes difficult and will eventually increase the maintenance cost and affect the evolution of the software.

2.2 Component-based Approach (Com-based)
Component-based approach views application system as a “composition of components” [16]. The components are viewed as “black-box” entity that can be used and reused to develop application systems. Component-based approach focuses more on “development of runtime component models which are targeted at the actual construction and deployment of systems” [16]. To achieve adaptation, component composition is modified dynamically to create variants. At any one time, the variant that best suited user requirements and operating conditions will be selected.

Most component-based approach uses component framework or infrastructure to implement adaptation. Among popular component framework used are CORBA, EJB, COM and DCOM. The framework or infrastructure will enable run-time reconfiguration of the component-based software thus, achieving desired adaptability. Component-frameworks use Interface Definition Language (IDL). IDL can lead to interoperability problem since it is not common for all component frameworks. COM, for example, uses Microsoft IDL (MIDL) which is different from IDL used by CORBA [17].

2.3 Middleware-based Approach (Mw-based)
Middleware can be defined as software that mask heterogeneity of operating environment and provide high-level programming abstraction to application developers [10]. Middleware-based approach attempts to implement software adaptation by segregating the development of application systems into different layers. Traditional middleware provides limited means to enable software adaptation. Enhancement of traditional middleware to address software adaptation has given birth to adaptive middleware. Adaptive middleware can be defined as middleware with the capability to dynamically change its functional behavior due to changes in its operating environments or user requirements [18].

Computational reflection and Aspect Oriented Programming (AOP) are two popular approaches in implementing adaptive middleware. Computational reflection allows the middleware to monitor its behavior and adapt itself dynamically. These properties make computational reflection important and efficient in providing adaptability for a highly dynamic system [19]. An example of adaptive middleware that implement this technique can be found in the work by [3, 20, 21]. AOP allows separation of cross-cutting concerns and implement each concern in a single unit called aspect. During compilation or executing, these aspects can be selectively woven together resulting in programs with new behavior [10]. Some examples of adaptive middleware that implement AOP can be found in [18, 22].

An advantage of middleware-based approach is it enables separation between functional behavior and adaptation behavior [21].

2.4 Agent-based Approach (Ag-based)
Agent-based approach is an area in Artificial Intelligence used to build intelligent software [23]. In [24], a software agent is defined as a software entity that is independent and able to behave autonomously. In the same literature, multi-agent system is referred to as system that consists of a set of software agents that interact with each other to reach common objectives. In [25], software agent is viewed as a computational entity with predetermined goals to be fulfilled.

Agent-based approach is used by a number of researchers to achieve software adaptation. Related works can be found in [25, 26]. Software agent can be used to enable software adaptation by acting as adaptation mediator in providing externalized adaptation mechanism. This approach is adopted by [25] using agent-based middleware called AMWare.

Agent-based approach uses higher level of abstraction than Object-Oriented approach, thus require different kind of thinking paradigm for developers to develop agent-based software. Novice developer will not get many benefits from Agent-based approach since it requires change of paradigm in developing software [27]. To achieve useful works or goals, a group of agents must cooperate and communicate efficiently. Communication between agents depends on asynchronous messaging; therefore, there are higher possibility of communication latency in agent-based system that will affect performance and latency of the system [28].

3. EVALUATION CRITERIA
The four approaches to software adaptation described in previous section will be compared based on selected evaluation criteria. The evaluation criteria were selected after a thorough literature
review on the topics of controlling software evolution. Software evolution occurs due to changes subjected to the software. Therefore, controlling negative effects of change will simplify software evolution.

Changes to software can be associated with changes in functional or non-functional requirements [29]. The five selected criteria are closely related to non-functional requirements of software. The following sub-sections briefly described the five criteria.

3.1 Scalability
Scalability refers to the “capacity of software to handle increasing loads or demand by users” [30]. This can be achieved by adding or removing system functionalities. Software needs to be scalable as its users grow in number. Any approach to software adaptation should consider scalability aspect to ensure the software developed will not be prematurely aged. Thus, scalability is important for adaptive software to prevent a difficult software evolution [31].

3.2 Context-awareness
Context awareness refers to ability of the software systems to be aware of its physical environment such as device change or device failure, and to adapt accordingly [11, 32]. Nowadays, context-awareness property is important in the development of software systems since it is common for the systems use external devices such as printers, biometric devices and many others. Failure of these devices will affect the usability of the software. Therefore, it is important for the software to dynamically adapt in the events of device failure or changes of devices. In this paper the context-aware ability of approaches under study is measured by investigating the ease of implementing context awareness capability.

3.3 Performance
Performance refers to acceptable response time of the system as perceived by the end-users. According to Laws of Software Evolution coined by Lehman [33], quality properties such as performance will degrade over time due to software evolution. As such, performance issues must be considered very early in the development life-cycle. Furthermore, performance is important in developing adaptive software since it must be balanced with cost-effectiveness [8]. There is an overhead cost in making a software adaptive [34]. It is not within the scope of this paper to perform quantitative performance analysis on each approach. Instead, at which level of the software lifecycle the performance of the system can be specified is investigated. The earlier the performance issue is addressed; the chance of creating a software system with acceptable performance with lower cost is higher.

3.4 Heterogeneity
Heterogeneity refers to the ability of the adaptive system to be executed in heterogeneous platforms. This criterion is very important since software systems may be implemented using different platforms and they might need to collaborate [35]. Furthermore, software that is developed on one platform should be able to be used on other platform with minimum change if required. As such, depending on certain proprietary framework for execution of software may affect its heterogeneity. Adaptive software must be developed with heterogeneity criterion in mind to ensure its adaptability to various operating platforms.

3.5 Dynamic Evolvability
Dynamic Evolvability refers to the ability of the approaches to create software systems that can respond to changes at run time. This criterion is closely related to dynamic adaptation where adaptation occurs at execution time instead of compile time [36]. This criterion is important since adaptive system must be able to respond to changes in its environment even during execution time without the need to interrupt the execution [37].

4. THE COMPARATIVE EVALUATION RESULT
This section describes the comparative evaluation result of the four approaches to software adaptation.

4.1 Scalability
Architecture-based approach scores High for scalability since concern for scalability can be captured earlier, i.e. during the analysis part. This is due to the reason that software architecture provides a big picture of the whole system. One of the artefacts during analysis phase is the high-level architecture model. During the development of high-level architecture model, concerns on the scalability can be included. In the design phase, the architecture model will be further refined and since scalability has been address in the earlier phase, it will be translated into design models. However, architecture-based approach does not address on how adaptive systems are to be constructed. Therefore, during implementation phase, it is up to other software construction mechanism to implement scalability as modelled by architecture-based approach.

Component-based approach scores medium since the scalability aspect is handled by component framework and this might differ from one framework to another. COM and JavaBeans for example, can only be hosted on single machine, therefore scalability is affected.

Middleware approach is highly scalable. In this approach, codes are segregated into middleware layer and application layer. The codes can be implemented across different machines to increase scalability.

Agent-oriented approach is highly scalable since software agents are autonomous. These agents can be distributed to different hosts to ensure scalability. In this approach, scalability can address at run-time due to the autonomous nature of software agents.

4.2 Context-awareness
In terms of context-awareness, all approaches except for Architecture-based score High. Architecture-based approach does not specify how this property can be constructed, therefore we cannot evaluate on how easy context-awareness capability can be implemented. Component-based and middleware based approaches model context-awareness during design phase. This information will be used during the implementation of the codes. However, context-awareness property is not generic to all type of systems. The concern for context-awareness is prevalent in the
4.3 Performance
Agent-based approach scores Low for this criteria. System performance is a challenging issue in Agent-based approach. Since agents must communicate together to achieve the goal of a system, the communication among them must be carefully designed and implemented. Bad design will result in bad implementation. This will lead to increasing latency and will affect system performance [28]. This risk is factored in for the evaluation; therefore, Agent-based approach is rated as Low for performance.

In Architecture-based approach, this criterion is rated as High. Architecture-based approach enables information regarding performance to be captured and expressed during the analysis phase and will be refined in subsequent phases. As architecture model specify relationship between system components and their connectivity, ability to capture performance requirement earlier in the life-cycle holds tremendous benefits. Design decisions can be made more accurate since system-level properties and constrain can be exposed earlier. This will help the designer to decide, for example, on load-balancing strategy to ensure better performance of the system.

In component-based and middleware-based approach, the performance of the system can be specified during the design and implementation phases. This offer lesser advantage as compared with architecture approaches that able to capture performance properties at a higher abstraction level. Hence, performance for these two approaches is rated as Medium.

4.4 Heterogeneity
Architecture-based approach presents an independent view of the systems, not restricted to any particular operating system or environment. Therefore, it allows for the development of adaptive system that can be executed in heterogeneous environment. Thus, it has a High heterogeneity level.

Component-based approach scores lowest among the three since the implementation is dependent on the component-framework used. If the adaptive system is developed on EJB component-framework, it cannot be used in COM-based framework. However, CORBA based component framework is claimed to be cross-platform and cross-language [36].

Middleware-based approach scores High because it is not restricted by any particular framework. The middleware can be developed to be “open” across platforms.

Some Agent-based implementation uses specific framework such as Java Agent Development Framework (JADE), therefore it becomes restricted to the availability of the framework [38]. Other implementation uses web service approach [9], which is more open thus increasing the heterogeneity level. Due to this reason, the score for Agent-based approach is Medium.

4.5 Dynamic Evolvability
All four approaches scores High in this criterion since they provide mechanism to enable dynamic adaptation of a software system.

The following table shows a simplified view of the comparison between the four selected approaches.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Arc-based</th>
<th>Com-based</th>
<th>Mw-based</th>
<th>Ag-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Context-awareness</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Performance</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Heterogeneity</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Dynamic Evolvability</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

4.6 The Proposed Framework
Our research direction is to develop adaptive framework as an approach to simplify software evolution. The framework will improve existing approaches to software adaptation.

Based on the result of the evaluation criteria, we found out that Middleware-based approach is the best in overall score. Therefore, the proposed framework will be implemented using Middleware-based approach. However, in terms of performance, Architecture-based approach fare the best. Due to this, some aspects of Architecture-based approach that is related to performance will be researched further and will be implemented in the framework.

The following figure (Figure 1), depicts the context diagram of the proposed framework.
In Figure 1, Application Layer refers to a layer where different types of application systems reside. Application systems can be of Smart Client, Thin Client or Thick Client type. These application systems will issue requests to use device services provided by the Device Layer.

Middleware Layer is where the proposed framework will be implemented. This layer provides two interfaces, one for communicating with the systems at the Application Layer and another interface is for communicating with the Device Layer.

Device Layer in the above diagram represents actual devices in the system. The devices usually come with their own device drivers and a set of API for device integration which can be proprietary or industry standards. In the above diagram, this interface is referred to as Physical Device Interface (P. Dev Int.). These API will be used by the Middleware Layer to integrate with devices and expose the functionalities of devices using web services.

In a general flow, applications that reside at the Application Layer will send requests to the framework in order to use services from the Device Layer. Certain components of the framework will entertain this request and finally, services from the Device Layer can be used by the requestor. The proposed framework has an important role of making sure that failure or changes at the device layer will not affect the requestor.

There are five main components of the framework namely Service Manager, Repository Manager, Adaptation Manager, Context Monitor, Device Interface and Protocol Adaptors. Brief explanations of the main components are as follows:-

- **Service Manager.** Service Manager is the first point of contact between Application Layer and it consists of a multithreaded component called Listener. Listener will entertain requests for service and put the requests into a Request Queue. Requests are made by applications that reside in the Application Layer to use services provided by the Device Layer. Once a request is put into the queue, the queue engine will trigger Repository Manager.

- **Repository Manager.** Repository Manager has the role of searching the information (in WSDL format) on the required web service registered in UDDI and put the result into the Reply Queue.

- **Adaptation Manager.** Adaptation Manager consists of two main components, Adaptation Planner and Adaptation Configurator. Adaptation Planner has the role of acquiring information regarding the changing environment. Adaptation Planner will choose the right plan from the Plan Repository. Adaptation Configurator receives the current plans for all devices and stores them in the Active Plan repository. Another role of Adaptation Configurator is to host adaptors to devices so that they can be dynamically load by the web services when required, for example, when new devices are added to the environment. Adaptation Configurator will also shield the different formats of device data and command from the web services by providing a common interface.

- **Context Monitor.** Context Monitor’s role is to monitor operating environment, checking for malfunction device or addition of new devices. Information on any changes to the operating environment will be put into the Context Queue. Trigger mechanism will be implemented so that whenever message is put into the queue, Adaptation Planner will be executed to evaluate changes and retrieved the correct plan.

- **Device Adapter.** This component is the actual interface to the API provided by device vendors and will be dynamically loaded when required.

The following Figure.2 is the high level view of the framework.
services that provide device services can be hosted on different machines to increase scalability.

Context-awareness criterion is handled by Context Monitor. This enables changes to the environment is detected and proper action can taken in order for the framework to adapt to the new changes. Due to this, the calling application will be shielded from changes that occur in operating environment.

Performance issue can be arrested earlier in the development of the framework since some parts of the Architecture-based approach will be adopted. Further research will be carried out in this area.

Heterogeneity is ensured since Middleware-based approach is selected to implement the proposed framework. No other third party software framework is needed for run-time execution. The use of web service also increases heterogeneity of the framework.

The proposed framework supports dynamic evolvability since web service approach enables dynamic binding of services to the calling applications.

4.7 Conclusion and Further Work

Software evolution may result in huge problem to an organization if it is not well planned. One of the ways to reduce the effect of software evolution is via software adaptation. Many approaches to software adaptation exist. Based on exhaustive research, we classify four main approaches to software adaptation namely Architecture-based, Component-based, Middleware-based and Agent-based. Five evaluation criteria were proposed for the purpose of systematically comparing the four approaches to software adaptation. Based on the result of the comparative evaluation, Middleware-based approach is found to be the best in the overall score. Architecture-based approach has an upper-hand in terms of the ability to specify adaptation needs very early in the software development life cycle, thus increasing the chance of developing software with better performance. We proposed a framework, using Middleware-based approach in order to simplify software evolution. The five evaluation criteria namely, scalability, context-awareness, performance, heterogeneity and dynamic-evolvability is used as prominent features of the proposed framework.

Towards the end of the research, this framework will be validated using industrial strength case studies. The framework will be further refined based on the result gathered from the validation. We also plan to develop automated tools that may support the proposed framework. The automated tools may be integrated with existing meta-CASE or commercial CASE tools. The proposed framework will contribute towards simplifying software evolution. As a result, more resilient software system can be developed and therefore maintenance efforts can be reduced.

5. REFERENCES


