Transaction-Aware Aspects with TransJ: An Initial Empirical Study to Demonstrate Improvement in Reusability

Anas M.R. AlSobeh
Computer Information Systems-Yarmouk University
Irbid, Jordan
Email: anas.alsobeh@yu.edu.jo

Stephen W. Clyde
Computer Science-Utah State University
Logan, Utah, USA
Email: stephen.clyde@usu.edu

***Abstract*—TransJ is an extension to AspectJ for encapsulating transaction-related cross-cutting concerns in modular aspects. This paper presents an empirical study to evaluate the reusability and performance cross-cutting concerns implemented with TransJ compare to AspectJ alone. As part this study, we define a reuse and performance quality model as an extension to an existing quality model. We then formalize eight hypotheses that can be tested using metrics from the quality model. Finally, to assess the hypotheses, we compare implementations of different sample applications across two study groups: one for TransJ and another for AspectJ. Results from the study show improvement in reusability when using TransJ, while preserving the performance.**

Keywords-Transaction-related Aspects; Aspect-Oriented Programming (AOPL); Abstractions; Transaction Joinpoint; Dynamic Weaving; Pointcuts; Transaction-related Contexts (TCC's); software reuse; and performance.

I. INTRODUCTION

The implementation of complex applications using Aspect-Oriented Software Development (AOSD) ―as a modern modularization technique― results in a better implementation structure relative to essential application qualities, such as maintainability, reusability, modularity, and reduce complexity [1]. One of the recognized strengths of Aspect-Oriented Programming Languages is the separation of concerns (SoC's) through the definition of modular abstractions, called Aspects, that reduce scattering and tangling of crosscutting concerns (CC's) in the application code. By definition, CC's impact multiple components of an application's core code. Common examples include logging, enforcement of real-time constraints, concurrency controls, transaction management, access controls, and so on. Implementing these such concerns directly into a Distributed Transaction Processing System (DTPS) can cause the scattering and tangling of code and, thus, make the code unnecessarily complex and difficult to understand, reuse, maintain, and evolve.

AspectJ is considered the de facto standard and the most widely used Aspect-Oriented Programming (AOP) framework for modeling CC's. It extends Java with mechanisms for supporting logic related to CC's, starting with aspect, which are first-class programming constructs for CC's [2][3]. Aspects encapsulate advice, pointcuts, and type-introduction declarations. An advice is a method that embodies some piece of CC functionality, but it is not called explicitly like class or object methods. Instead the execution of an advice method is woven into the core application according to specifications, called pointcuts. A pointcut is a predicate that defines where to weave advice at compile time and when to execute at runtime. More specifically, it is a pattern that identifies a set of joinpoints, which are best characterized as intervals within the program's execution flow. A joinpoint represents places (intervals or times) in execution on program and advice run before, after, or around these intervals [2].

AspectJ supports many different kinds of joinpoints, such as fields, methods, constructors, and catch blocks in exception handling, but they only related to program-language abstractions and their contexts are limited to single-threaded execution flows. The problem is that AspectJ does not inherently handle higher level abstractions or application-level contexts, like transactions, which may be tied to runtime objects and used by multiple execution threads or processes. Hence, AspectJ cannot directly support the dynamic weaving of advice into transaction abstractions or directly leverage transaction context information.

TransJ is an extension to AspectJ that introduces transaction-aware aspects, independent of any specific transaction-processing framework. With TransJ, developers can weave Transaction-Related Crosscutting Concerns (TCC's) into a DTPS in a modular and reusable way, while preserving core functionality, and obliviousness to those TCC's. (See Section II).

In this paper, we report on a study that investigated the impact of TransJ on the reuse of DTPS code while preserving performance. It does so by evaluating certain desirable characteristics and attributes defined in an extended quality model (see Section III) using a set of computable metrics. Based on an initial theoretically investigation, we hypothesized that developers would see improvement reuse improvements while preserving the software performance when using TransJ. We formalize this notion into eight specific hypotheses (see Section IV). Section V explains our experiment methodology; selection of the sample software application; and identification of interesting TCCs that would provide good coverage. The methodology also included supporting activities such as recruitment and training of the developers as test subjects. After the experiment, we collected and analyzed data from the code, journals, questionnaires, and surveys. From the results (see Section VI) of the study, we conclude that application using TransJ have less coupling (less scattered), less complex, and required less effort and time to enhance. Also, they are more cohesive (less tangling) and oblivious without sacrificing the performance. These preliminary results lead us to believe that further experimentation with TransJ and refinement of its framework could prove to be very beneficial to a wide range of software applications.

II. HIGH-LEVEL OVERVIEW OF TRANSJ

Fig. 1 provides a high-level overview the TransJ's layered design [6], in which each layer embodies reusable functionality and provides services to the layer above it and uses the services of the layer below it.

One component at the lowest layer is the Unified Model for Joinpoints in Distributed Transactions (UMJDT), first introduced in a 2014 ICSEA paper [5]. The UMJDT is a conceptual model for weaving advice into distributed transactions that captures key events and context information, and use that ideas to define interesting joinpoints relative to transaction execution and context data for woven advice.

AspectJ and some transaction-processing framework, like JTA, are two components at the lowest level.

 ISBN: 978-1-61208-498-5

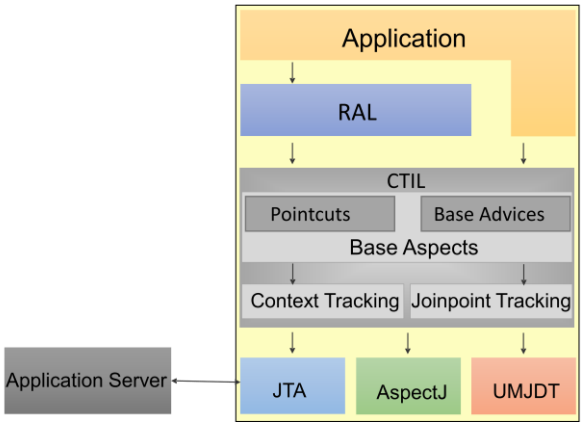

Figure 1. TransJ Architectural Block Diagram.

The Core TransJ Infrastructure Layer (CTIL) is a library that implements a transaction joinpoint model on top of an AspectJ joinpoint model. It defines transaction abstractions, transaction-events joinpoints, a collection of pointcuts for gathering context information that can be used in the advice code, and mechanisms to track transaction contexts and joinpoints. This library allows developers to treat transactions as first-class concepts into which aspects can be woven, promoting greater enhancements, obliviousness, and localization, along with code reusability.

The Reusable Aspect Layer (RAL) is a toolkit-like collection of transaction-related aspects that application programmers should find useful in many different kinds of applications with significant transaction requirements. These reusable aspects can decrease the development time; make CC's more understandable, reusable, and predictable; and ensure that the core application is oblivious to the CC's.

Application-level Aspect Layer is where application developers implement transaction-related aspects using the abstractions provided by TransJ directly or by specializing the aspects from the RAL. These aspects can encapsulate complex TCC behaviors in understandable, predictable and reusable software components, without sacrificing obliviousness or efficiency [6].

III. EXTENDED-QUALITY MODEL FOR TRANSACTIONAL APPLICATION (EQMTA)

Many empirical studies have found that different soft-ware factors influence the quality of a software system [7][8][9]. Of these, we picked reusability and performance as important qualities to consider initially because of potential for cost savings that they both represent. To formalize the reuse and performance qualities, we adapt and extend the Extended-Quality Model [9], which was based on the Comparison Quality Metrics (Sant'Anna quality model) [1][7] to include quality factors and internal attributes specific for DTPS's, forming EQMTA.

EQMTA consists of four elements: Qualities, Factors, Quality Attributes, and Metrics. The qualities, i.e., reusability and performance, are the most abstract concepts in the model and represent the ultimate goals of "good" software. Each quality is affected by one or more factors, which are in turn determined by quality attributes (internal attributes). The quality attributes describe the internal view of the system attributes with a set of quality metrics that are de-fined and used to provide a scale and method for measurement.

Fig. 2 shows the specific qualities, quality factors, and quality attributes of the EQMTA's suite, and Fig. 3 shows the metrics. A single star (*) next to an element in either of these figures tags a concept that not exist in the original EQM [8] or Comparison Quality Model [1]. Double stars (**) mark elements that are in the previous

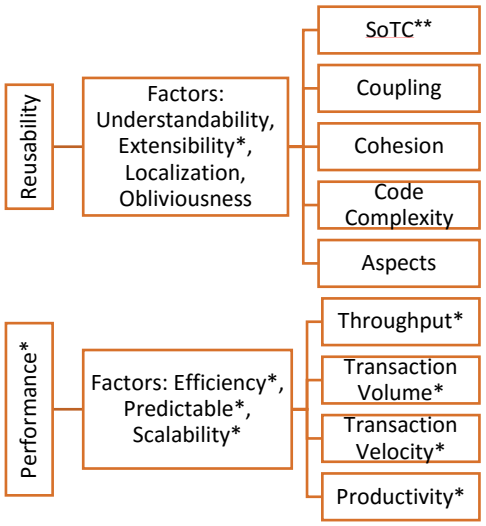

Figure 2. Extended-Quality Model for Transactional Applications (EQMTA)

models, but have been modified to be a measure quality in transaction systems.

The quality factors are the secondary quality attributes that influence the defining primary qualities and associated with well-established internal quality attributes of the soft-ware systems as shown in Fig. 2. Raza [8] proposes three important characteristics of modular code, namely understandable, obliviousness and localization of design decisions. Hence, reasoning reusability in terms of understandability, localization of design decisions, and obliviousness are not complete. Introduction of efficiency, predictability, and scalability are also equally important. At the time Parnas [11] and Coady [12] proposed that the definition of reusable modular code, obliviousness and extensibility has not been documented as fundamental design principles. How-ever, in the context of our research experiment, they are critical to understanding the impact of TransJ.

A. EQMTA Metrics

The EQMTA contains 29 design and code metrics for the 9 internal attributes shown in Fig. 3. In some cases, we had to adapt the metrics to better evaluate the attributes in DTPS. Twelve of the metrics can be computed automatically from the code written by the subjects. The others have to be computed by hand. Below are brief descriptions of these metrics, so the reader can better understand the

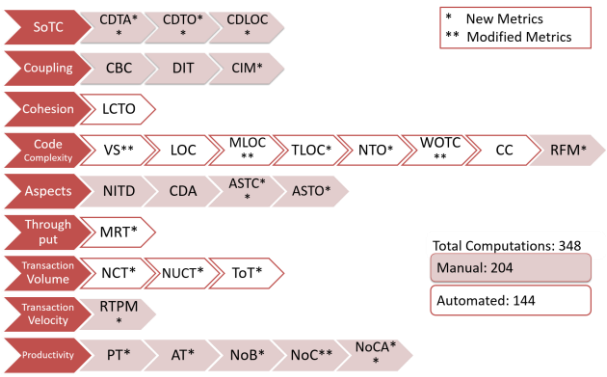

Figure 3. Measurement Metrics in EQMTA

results presented in Section VI. For space considerations, the full definitions of all metrics are not shown.

1) Separation of Transaction Concern (SoTC)/ Scattering Metrics

SoTC defines the ability to capture, encapsulate and manipulate unnecessary complexities of transaction system that are relevant to a particular concern [13]. The Concern Diffusion in Transaction Application (CDTA), Concern Diffusion over Transaction Operations (CDTO) and Concern Diffusion over Line of Code (CDLOC) are SoTC metrics. CDTA number of primary transaction components (class or aspect) whose main purpose is to contribute to the implementation of a concern. CDTO counts the number of methods and advices that access any primary transaction component to pull all relevant operation context information by calling their methods or using them in formal parameters, local variables, return types, and throws declarations. Constructors also are counted as operations. CDLOC counts the total lines of primary transaction components whose main purpose is to contribute to the implementation of a single transaction-related concern.

2) Transaction-related Coupling Metrics

It is an indication of the strength of interconnections between the transaction components in a DTPS [10][14]. Coupling between Components (CBC), Depth Inheritance Tree (DIT), and Coupling on Intercepted Modules (CIM) are coupling metrics. CIM counts the number of classes, aspects or interfaces explicitly named in pointcuts of a given aspect. High values of these metrics indicate tight coupling, due to high crosscutting.

3) Transaction-related Cohesion and Tangling Metrics

The cohesion of a transaction is a measure of the degree fitness between its internal pieces [7]. Lack of Cohesion in Transaction Operations (LCTO) measures the lack of cohesion of a class or aspect in terms of the occurrences of the method and advice pairs that do not access the same context variable and hence should be reasonably separated [15]. High cohesion often correlates with loose coupling, and vice versa [10]. Low coupling is often an indicator of a well-structured DTPS and a good design, and when combined with high cohesion, supports the general goals of high reusability.

4) Complexity Metrics

The EQMTA defines metrics that are concerned with the different aspects of the DTPS complexity. It measures how transaction components are structurally interrelated to one another and measures the size of a software system's design and code [1]. In EQMTA, the Vocabulary Size (VS), Line of Code (LOC), Method Lines of Code (MLOC), Transaction Lines of Code (TLOC), Number of Transaction Operations (NTO), and Weighted Operations per Transaction Component (WOTC), McCabe's Cyclomatic Complexity (CC), and Response for Module (RFM) are complexity and size metrics in EQMTA. VS counts the number of classes and aspects into the DTPS. Sant' Anna mentioned that if the number of components increases, it is a clue of more cohesive and less tangled set of abstract datatype concepts [1]. NTO counts the number of transaction-related operations. A transaction contains with more operations are less likely to be reused and assumed to have more complex collaboration with other components. Sometimes LOC is less, but NTO is more, which indicates that the transaction component is more complex. The number of advices and methods and complexity is an indication of how much time and effort is required to develop and maintain the transaction-related components. The larger the value of WOTC, the more complex the program would be [15][16]. CC is intended to measure system complexity by examining the software program's flow graph [17]. In practice, CC amounts to a count of the decision points present in the software system. The high value of CC affects transaction components reuse. RFM counts the number of methods and advices that are executed by a given transaction in response to the request received by another transaction or system. Transactions with a higher RFM value are more complex and complicated.

5) Aspects/Obliviousness Metrics

The EQMTA involves metrics on concerns that evolve into concrete pieces of code, i.e., Aspects, and contribute directly to the core functionality of the transaction software system [8]. This model defines the following aspect metrics: Number of Inter-type Declarations (NITD), Crosscutting Degree of an Aspect (CDA), Aspect Scattering over Transaction Components (ASTC), and Aspect Scattering over Transaction Operations (ASTO).

6) Transaction Throughput Metrics

Transaction throughput is the rate at which transactions are processed by the system. The EQMTA defines the rate of the Mean Response Time (MRT) to measure the performance of an individual transaction, in milliseconds. MRT represents the amount of time required for transaction completion, i.e., commit or abort. The response time for a transaction tends to decrease as you increase overall throughput.

7) Transaction Volume Metrics

Transaction volume is an indication of the efficiency of transaction system to handle huge data volume, which determine the amount of transactions processed by the system over the defined period of time, i.e., second. The EQMTA defines the following transaction volume metrics: Number of the Committed Transactions (NCT), Number of the uncommitted (aborted) Transactions (NUCT), and Timed-out Transaction (ToT).

8) Transaction Velocity Metrics

Transaction Velocity gives an indication of the performance of the transaction system. Rate of the Transaction Per Minute (RTPM) is the only velocity metric in EQMTA that measures velocity of a transaction in our model. RTPM is the average number of transactions that are begin completed, either committed, aborted, or timed-out, per minute on the transaction system.

9) Productivity Metrics

Productivity is a measure of the amount of effort needed to understand, implement and debug the transaction system components. It considers the amount of bugs, and total development time into active and passive times. Active Time (AT), Passive Times (PT), Number of Bugs (NoB), Number of Changes in Concern at the application level (NoC), and Number of Changes in Concern and its Application (NoCA) are productivity metrics. NoC and NoCA count the number of changes required to reuse the concern for another application, and to maintain the concern, respectively. The difference among them is that the NoC only considers changes in the concern; however, the NoCA considers changes both in the concern and application. A lower value of PT, AT, NoB, NoC and NoCA is more desired to increase the efficiency of the development transaction-related components.

IV. EXPERIMENTAL HYPOTHESES

The theoretical ideas underpinning TransJ lead to the following eight hypotheses. All of these hypotheses have the same premise and are tested using the EQTMA metrics. Let S represent a software system that has TCC's and is implemented in AspectJ. Also, let S' be an implementation of same system using TransJ. The premise is implementation of the TCC in S' make reasonably effective use of TransJ.

A. S' has better encapsulation and Separation of Concerns (SoCs) and less scattering than S.
B. S' has a lower coupling than S.
C. S' has higher cohesion and less tangling than S.
D. S' not significantly larger or complex S.
E. S' is significantly more oblivious to TCC's than S.

F. The runtime of S' is no worse than S
G. The implementation TCC's in S' requires a smaller number of changes to reuse compared to S.
H. The total programming hours for S' is less than S, indicating that S' is less complex and more readable than S.

V. EXPERIMENTAL PROCEDURE

The research experiment consisted of the following steps:

1. Experiment Approval: We submitted an application for conducting this Human Research Experiment to the USU IRB and got its approval [4]. Before submitting this application, all the researchers passed the online human research experiment-training course offered through the Collaborative Institutional Training Initiative (CITI) [18].
2. Selection of Applications: we developed three non-trivial software applications that were diverse in the way they implemented transactions; used JTA API, X/Open standards, Jboss Application Server; multithreaded; and therefore provide a good coverage of different types of distributed transactions as shown in Table 1. We used Java 2 Enterprise Edition (J2EE) to build these non-trivial applications. They include classes for distributed resources and make used Enterprise Java Bean (EJB), Java Persistence (JPA), Maven, Hibernate, Jboss, JTA, Arjuna, and MySql database drivers. The current EJB architecture supports flat transactions only, but the Arjuna supports nested transactions in the application. Most of the implementation details are not relevant to the contributions of this paper, and are there omitted for space considerations.
3. Selection of TCC's: we picked three common TCC's for the experience such that they were applied to all the sample applications and the various concepts of transactions, as shown in Table 2. To reduce chaos in our data, we wanted to make sure that these CC's were adequately simple to a novice developer could understand and integrate them into the selected sample applications in less than 15 hours, regardless of whether TransJ or AspectJ is used.
4. Recruitment of Developers: To transparently recruit the developers, we sent invitation letters and then recruited four developers who were experienced OO and AOP software development, Java, transaction, and software-engineering design principles such as reusability and performance. We randomly organized them into two study groups: 1 and 2. Group 1 implemented using an AOP approach and Group 2 used TransJ fashion. Next, the participants completed a survey that assessed their background and skill levels. We also provided JTA, Arjuna library, Jboss, AOP training to developers in Groups, and had them worked through some practice applications. Similarly, we trained Group 2 developers with TransJ, and had them worked through some practice applications. Next, each developer filled a pre-implementation questionnaire, developed the application using initial requirements, recorded hourly journals and completed a post implementation questionnaire.
5. We analyzed the understanding of the requirements, familiarity with the language and tools, and debugging the most prominent challenges. They also recorded hourly journals of productivity. At the end of implementation, each developer filled the post-implementation questionnaire. Observation of this questionnaire indicated that all developers correctly understood the requirements, familiarized with the language, tools, and debugged the challenges.

   We measured EQMTA code metrics using both manual-based and automated tool-based methods [19][20]. Total measurements include following: experiment input variables included a total of four developers and three applications with each; experiment generated a total of 12 software systems against which the metrics need to be applied; the 29 code metrics of EQMTA, which will have a total of 348 measurements. Of these, 144 measurements from 12 metrics were generated using tools, and 204 measurements from 17 metrics were calculated manually.

TABLE 1. CATEGORIES OF SELECTED APPLICATIONS

Applications		Gadget Manufacturing System	Conference Registration System	Local Bank System
1	Distributed	*	*	
	Local			*
2	Flat		*	*
	Nested	*		
3	Few Resources		*	*
	Many Resources	*		
4	Low Concurrency			*
	High Concurrency	*	*	
5	Low Potential for Conflict		*	*
	High Potential for Conflict	*		

TABLE 2. SELECTED TRANSACTION-RELATED CROSSCUTTING CONCERNS (TCC)

#	Aspect Name	Description
1	**Measuring Performance**	It measures some performance-related statistics for transaction-based applications between a client and server, such as turn-around time (i.e., response time).
2	**Data-Sharing Optimization**	It shares context information across hosts only when necessary.
3	**Audit Trail**	It records a history of actions executed by transactions and users in order to monitor transaction activities and provide assurance that meet the predefined minimum requirements.

VI. EXPERIMENT RESULTS

This section presents empirical results relevant to the eight hypotheses. We analyzed and evaluated the reusability and performance using the code developed by the student participants, questionnaires, hourly journals, and maintenance history. In the following graphs, the vertical axes represent the measurements, and the horizontal axes represent the three activities of the experiment. For each activity, there are two bars: a blue bar for the results of AspectJ group and an orange bar for the results of TransJ group. For space limitation, we did not show all results.

A. *S' has better encapsulation and Separation of Concerns (SoCs) and less scattering than S*

From the graphs in Fig. 4, we found that the interest average of CDTA, CDTO and CDLOC values for TransJ went to zero in all three activities of the experiment, and the result was significantly different from AspectJ in the all activities. The reason for this phenomenon is that TransJ pointcuts provide total obliviousness between the transaction application and TCC's. AspectJ, transaction

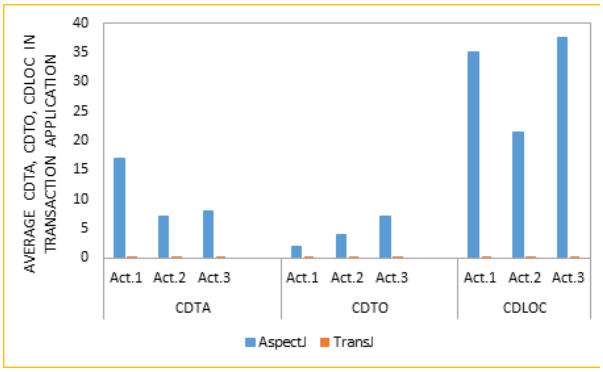

Figure 4. CDTA, CDTO, and CDLOC Coverage over Applications

components and their operations for CC's were significantly more diffused in the transaction application because the pointcuts had to be tied to programming constructs instead of transaction abstractions. From these results, we can conclude that the first hypothesis holds true for better separation of concerns in TransJ than in AspectJ.

B. $\mathcal{S}'$ has a lower coupling than $\mathcal{S}$

Fig. 5 shows that TransJ implementation decreased the values of CBC, DIT, and CIM in all the three activities of the experiment. TransJ removed dependencies and did not maintain any direct relationship between TCC's and the core transaction application components. In AspectJ, unnecessary coupling of TCC's with the core application components increased CBC, which hindered reuse and code understandability.

On the one hand, wide variations were found in DIT and CIM metrics from TransJ group and AspectJ group. The most significant indicator of the decrease in coupling between aspects and the core code is the impact of TransJ's joinpoints on the CIM metric. This metric counts the number of modules explicitly named in pointcuts. Compared to the AspectJ activities, the TransJ activities have a reduction of 100%, 100% and 100% in CIM (i.e., all of the three activities have an average value of zero for CIM metric). This was caused by providing a comprehensive set of pointcuts, which fully encapsulates the distributed transaction abstractions. This allows participant programmers to reuse the pointcuts directly, so they did not need to override or inherit the aspect components to name in the pointcuts of a given class. In contrast, the AspectJ programmers suffered from a lack of clarity of relationship among TCC's and application components, wherein aspects acquire context information from one of more classes. Thus, they preferred to inherit all of the attributes and operations from parent (superclass) methods in CC's to share context data across aspects and distributed transaction application components.

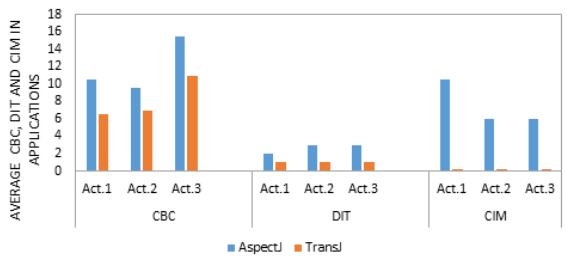

Figure 5. CBC, DIT and CIM Coverage over Applications

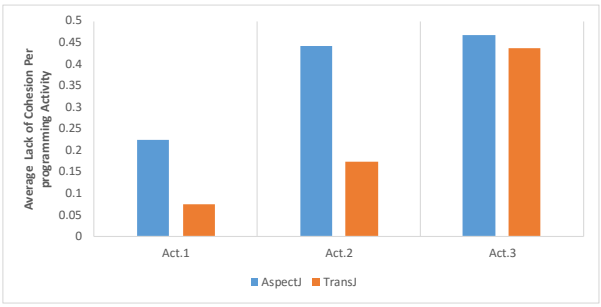

Figure 6. LCTO Coverage over Applications

In consonance with these results, we can confidently conclude that the second hypothesis holds true for reduced coupling in TransJ compared with AspectJ.

C. $\mathcal{S}'$ has higher cohesion and less tangling than $\mathcal{S}$

In Fig. 6, the result reveals that TransJ maintains a lower value for LCTO than AspectJ in all the three activities of the experiment. Thus, TransJ promoted encapsulation with implementing a more independent component that implements a single logical function (more cohesive) than implemented with AspectJ. Compared to the AspectJ group, the TransJ group improved cohesion in all activities, sometimes significantly (from 8% to 75%). The decrease in the cohesion of the AspectJ activities is caused by the need to extract new methods to expose advisable joinpoints i.e., multiple transaction joinpoints cannot be advised as an atomic unit (e.g., begin – commit, begin – abort, or lock – release). From these results, we conclude that the third hypothesis holds true for increased cohesion in TransJ than in AspectJ.

D. $\mathcal{S}'$ not significantly larger or complex $\mathcal{S}$

Figures 7 (a) through 7 (e) show that TransJ implementations decreased the metric values for LOC, MLOC, TLOC, NOT, WOTC, CC and RFM and increased VS value in all the three activities of the experiment. In comparison with TransJ, AspectJ programmers found the aspects and application code tends to contain very terse pointcuts, advices and extra code, especially, when combined with transaction constructs, such as transaction demarcations, to pull all relevant context information. In TransJ, two induced factors affect these metrics: the UMJDT model captures various general distributed transaction abstractions in meaningful, reusable joinpoints and a set of base aspects, which help developers implement the TCC's in simpler and logical method bodies, i.e., advice, with no extra lines of codes and less number of operations and advices, thus this reduced the RFM value. Second, TransJ's joinpoints referenced by broad contexts and stable pointcut definitions, therefore, applications did not need additional context information, such as an identifier or lock snapshot. This allowed the reusable and application-level aspects to inherit or reuse pointcuts to apply the logic of TCC' in appropriate transaction places. Hence, TransJ reduced the values of MLOC, TLOC, NTO, WOTC, and RFM. Fig. 7 (d) shows that the value of CC is smaller for TransJ than AspectJ, because TransJ hides complex transaction abstractions, as mentioned, which result in simple conditional statements and less tangled code. As predicted by the above hypothesis, results shown in Fig. 7 (e) give sufficient evidence that the average VS value of all programs was more for TransJ than AspectJ, due to inlined code in transaction scopes being extracted and gathered to inner classes, i.e., contexts and base aspects (caused improvements of 12% to 23%). Although the number of components were more in TransJ implementations, but they were more cohesive. From these results, we can confidently conclude that

 ISBN: 978-1-61208-498-5

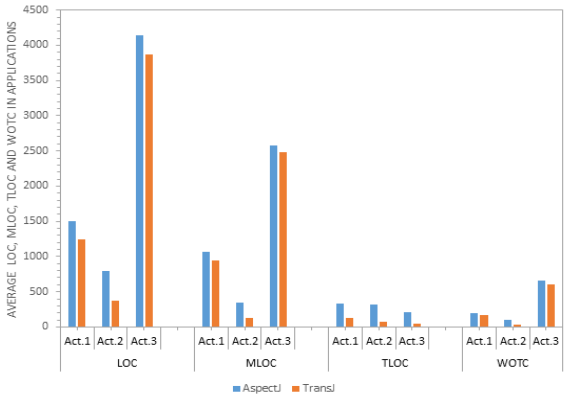

Figure 7 (a) Average LOC, MLOC, TLOC and WOTC over Applications

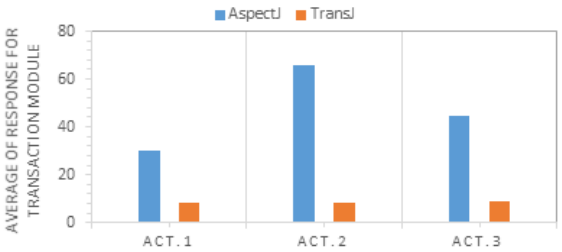

Figure 7 (b) Average RFM over Applications

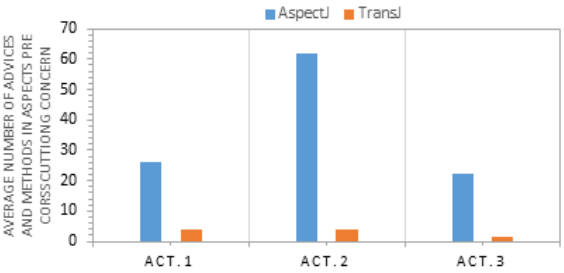

Figure 7 (c) Average NTO over Applications

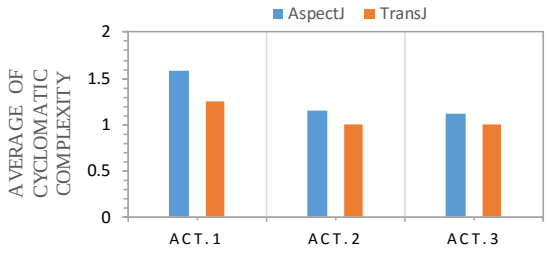

Figure 7 (d) Average CC over Applications

the fourth hypothesis hold true for less complex and a small code size software in TransJ compared with AspectJ.

E. *S′ is significantly more oblivious to TCC's than S*

Fig. 8 shows that TransJ implementations significantly reduced the values of NITD, CDA, ASTC and ASTO metrics. Compared to

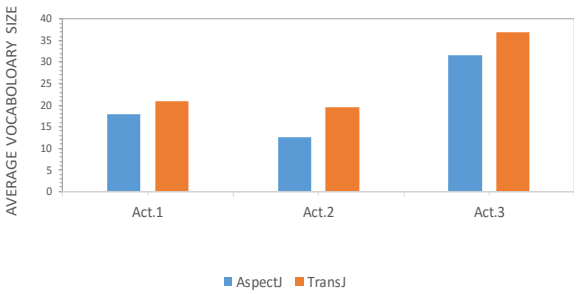

Figure 7 (e) Average VS over Applications

the AspectJ, NITD and CDA for all TransJ activities differed by 100%. The reason for having this result, i.e., zero value, TransJ programmers directly used transaction abstractions and did not need to use Inter-Type Declarations (ITDs) for sharing of context information between application and aspect components. Significant reduction in ASTC and ASTO was due to the layers of indirections among the transaction application and aspect components, which TransJ provides but are missing in AspectJ. In a nutshell, the improvement of the TransJ activities verse the AspectJ activities was caused by (a) the higher level of reuse of base aspects, and (b) scoped joinpoints, i.e., contexts, eliminating the need to create operations to expose new joinpoints. From these results, we can confidently conclude that the fifth hypothesis hold true for less oblivious software CC's in TransJ compared with AspectJ.

F. *The runtime of S′ is no worse than S*

Figures 9 (a) through 9 (c) show that TransJ implementation slightly decreased the metric values for MRT, NUCT, ToT, and slightly increased NCT with maintaining the RTPM in all three activities of the experiment. TransJ allows dynamic weaving of aspects at run-time by looking up to the contexts instead of needing to programing by hand as done in AspectJ. Figures 9 (a) and 9 (b) indicate that the TransJ group performed very slightly better than the AspectJ group for Act.1 and Act.2 with almost 0% improvement for Act.3. This lack of improvement for Act.3 was caused by the overhead of creating a transaction and transaction operation thread instances, synchronization and the high concurrent potential for conflicts over the shared resource. In other words, there are no major differences between the efficiency of TransJ activities and AspectJ activities.

Fig. 9 (c) shows that the results for the NUCT and ToT metrics remained the same for the Act.1 and Act.2. However, in Act.3 TransJ decreased very slightly the potential of having better ToT and NUCT values. The decrease in NUCT and ToT values in TransJ at Act.3 was caused by exposing advisable joinpoints, i.e., lockingJP and resourceLockedJP and dynamic weaving of aspects on them.

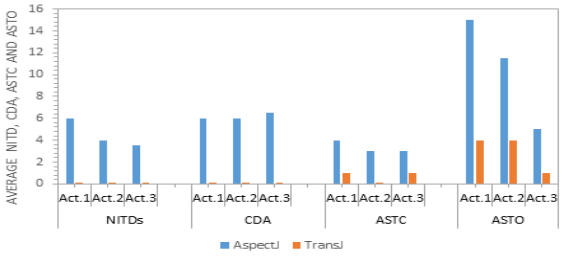

Figure 8. Average NITD, CDA, ASTC and ASTO over Applications

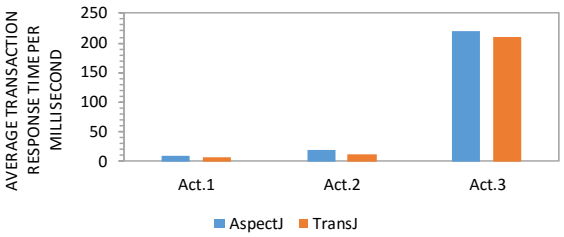

Figure 9 (a) Average MRT over Applications

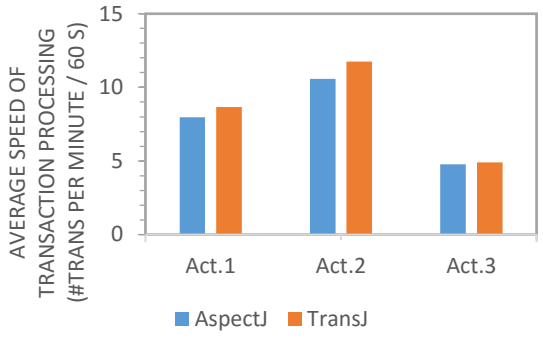

Figure 9 (b) Average Transaction Velocity (RTPM) over Applications

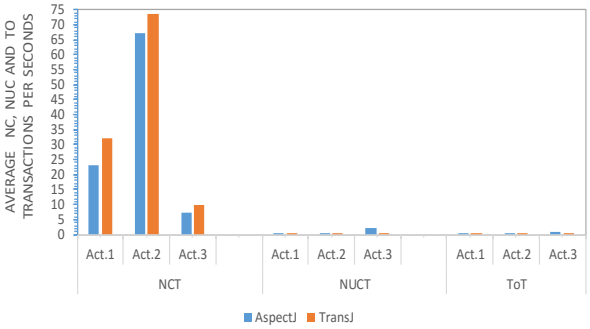

Figure 9 (c) Average NCT, NUCT and ToT over Applications

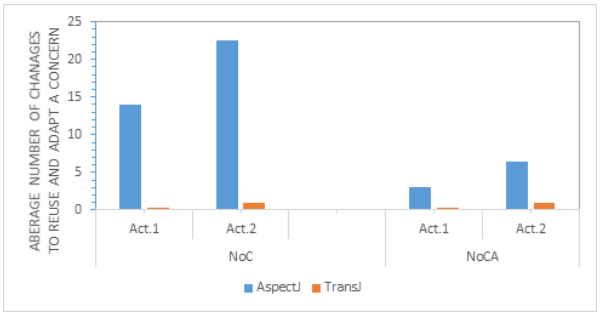

Figure 10 (a) Average Number of Changes of Performance Measurement Concern over Conference Registration System and Bank System Applications

These joinpoints represented an indication of the benefits that can come when concurrent operations access the shared resource. However, there are no major differences between the throughput of TransJ activities and AspectJ activities. In a nutshell, the results of figures do not give sufficient evidence to claim that the benefits of improving software performance. But from these results, we can confidently conclude that the sixth hypothesis holds true: preserving runtime performance in TransJ compared to AspectJ.

G. *The implementation TCC's in $\mathcal{S}'$ requires a smaller number of changes to reuse compared to $\mathcal{S}$*

From the results shown in Fig. 10 (a), we can see that TransJ implementation significantly reduced the changes required to reuse the performance measurement concern implementations in Act.1 and Act.2. This means that the application is more amenable to extension.

Compared with AspectJ, the presence of joinpoints in the base aspect of TransJ allows the implementation of the CC' logic in reusable and application-level aspects, which allow contexts and CC's to be explicitly communicated. Fig. 10 (a) presents the percentage of CC's that were implemented by abstract aspects (in base aspect). The data confirm that significant increases in reusability can be gained by applying TransJ's joinpoints where appropriate.

Fig. 10 (b) provides another graphical representation of the analysis of reuse for AspectJ and TransJ. The orange-colored graphs represent scattering in TransJ (aspects only) and the blue-colored graphs represent scattering in AspectJ implementations. The scat-

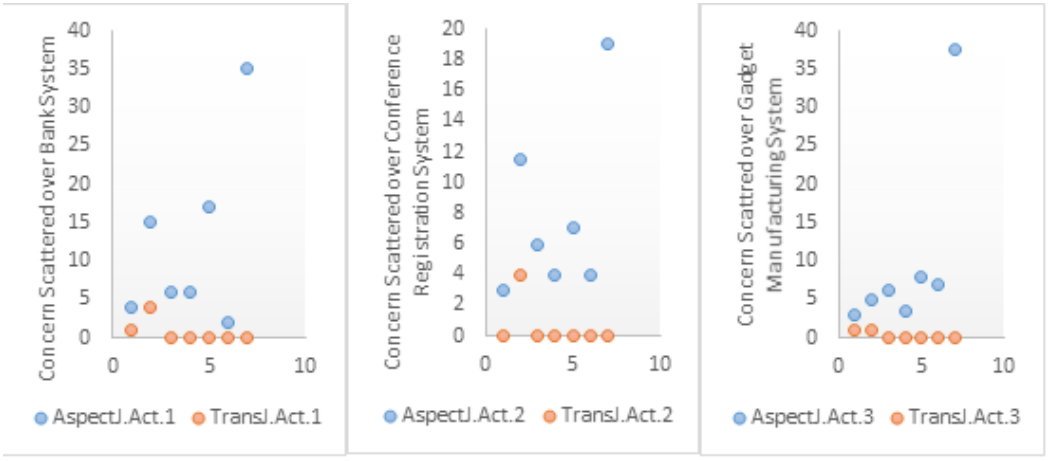

Figure 10 (b) ASTC, ASTO, CDA, NITD, CDTA, CDTO and CDLOC over Applications of AspectJ and TransJ

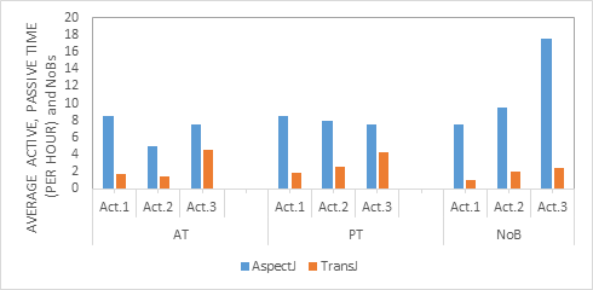

Figure 11. Average AT, PT and NoBs over Applications

tered points in the graph indicate that the number of changes required for reusing a concern with TransJ and AspectJ in different activities, respectively. The scattered points represent ASTC, ASTO, CDA, NITD, CDTA, CDTO, and CDLOC metrics results. Overall, activities of TransJ (highly reusable and more extensible), but were highly scattered for AspectJ. The reason for less scattering is discussed above. From these results, we can conclude that the seventh hypothesis holds true: more reusability and extensibility in TransJ compared to AspectJ.

H. *The total programming hours for $\mathcal{S}'$ is less than $\mathcal{S}$, indicating that $\mathcal{S}'$ is less complex and more readable than $\mathcal{S}$*

From the results shown in Fig. 11, we can see that TransJ significantly reduced the period that required to read, understand, implement, and debug the implementations of TCC's in all activities of the experiment compared to AspectJ. These results confirm that the applications were more flexible to implement with TransJ and were robust with respect to bugs and error compared to the AspectJ implementation. In addition, this figure indicates that TransJ participants performed significantly better than the AspectJ participants for all activities.

PT represents the amount of time they spent on reading the source code, understanding secondary requirements and looking for bugs. The increases in the PT in the AspectJ activities are caused by the need to study the whole code to find new pointcuts to expose advisable joinpoints and to gather the relevant information to a specific context that is required to weave the CC's of appropriate joinpoints. In contrast, TransJ provides pointcuts that help developers code the CC's obliviously. In addition, they do not need to create shared data structures, i.e., contexts, to have an explicit cooperation between base application code and aspects. This one simple benefit in the mindset of programmers can drastically reduce the number and seriousness of bugs, i.e., NoBs. From these results, we can confidently conclude that the eighth hypothesis hold true: less software development time is required for TransJ than for AspectJ.

VII. SUMMARY

In ICSEA 2014, we presented the new conceptual model, i.e., UMJDT, to define interesting joinpoints relative to transaction execution and context data for woven advice. TransJ is a new abstract framework, which allows developers to encapsulate TCC's in reusable and cohesive modules [6]. This paper presents a preliminary research experiment on hoped-for benefits of TransJ in comparison with AspectJ. It defines an extended-quality model for transactional application, then setup an experiment methodology, involving 8 hypotheses and data collection from 12 applications. Initial findings provide sufficient evidence to conclude that TransJ is capable of encapsulating a wide range of TCC's and that it can provide more modular, reusable distributed transaction software without sacrificing the performance. We hope to gather more empirical evidences of the TransJ's value by increasing the number of aspects in the reusable aspect library and by continuing to expand the number and types of applications that use TransJ. Our future research will include more formal software-engineering productivity experiments to verify the performance belief. TransJ can be extended for distributed remote pointcuts that would simplify the implementation of even more complex crosscutting concerns, such as recovery, or multithreaded in a distributed system.

REFERENCES

[1] C. Sant'Anna, A. F. Garcia, C. F. G. Chavez, C. J. de Lucena, and A. Staa, "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework," In Proc. 17th Brazilian Symp. Software Engineering, Manaus, Brazil, pp. 19-34, 2003, doi: PUCRioInf, MCC26/03.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.M. Loingter, and J. Irwin, "Aspect-Oriented Programming." Proceedings of ECOOP '97, Springer Verlag, pages 220--242, 1997.

[3] G. Kiczales and M. Mezini, "Aspect-Oriented Programming and Modular Reasoning," in ICSE 2005, pp. 49-58, 2005.

[4] Office of Research and Graduate Studies at Utah State University, Institutional Review Board [Online]. [retrieved: June, 2015], Available: http://rgs.usu.edu/irb/.

[5] A. AlSobeh and S. Clyde, "Unified Conceptual Model for Joinpoints in Distributed Transactions." ICSE'14. The Ninth International Conference on Software Engineering Advances. Nice, France. October, pp. 8-15, 2014, ISBN: 978-1-61208-367-4.

[6] A. AlSobeh, "Improving Reuse of Distributed Transaction Softwares with Transaction-aware Aspects," in Ph.D. Dissertation, Computer Science Department, Utah State University 2015. Paper 4590. http://digitalcommons.usu.edu/etd/4590

[7] C. Nunes, U. Kulesza, C. Sant'Anna, I. Nunes, and C. Lucena, "On the Modularity Assessment of Aspect-Oriented Multiagent Architectures: a Quantitative Study." International Journal of Agent-Oriented Software Engineering, v. 2, pp. 34- 61, 2008.

[8] A. Raza and S. Clyde, "Communication Aspects with CommJ: Initial Experiment Show Promising Improvements in Reusability and Maintainability," ICSEA'14, pp. 48-55, 2014, Nice, France, Oct. 2014.

[9] A. Raza and S. Clyde, "Weaving Crosscutting Concerns into Interprocess Communications (IPC) in AspectJ," ICSEA 2013. Venice, Italy, pp. 234-240, 2013, ISBN: 978-1-61208-304-9.

[10] R. Burrows, A. Garcia and F. Taiani, "Coupling Metrics for Aspect-Oriented Programming: A Systematic Review of Maintainability Studies," In Evaluation of Novel Approaches to Software Engineering, volume 69 of Communications in Computer and Information Science, pp. 277-290, 2010.

[11] L. Parnas, "On the Criteria to be used in Decomposing Systems into Modules," Commun. ACM, vol. 15, no.12, pp. 1053-1058, Dec. 1972.

[12] Y. Coady et al., "Can AOP Support Extensibility in Client-Server Architecture?" In European Conference on Object-Oriented Programming (ECOOP), Aspect-Oriented Programming Workshop, June 2001.

[13] C. Sant'Anna, E. Figueiredo, A. Garcia, and C. J. P. Lucena, "On the Modularity of Software Architectures: A Concern-Driven Measurement Framework." In Proc. ECSA, pp. 24-26, 2007, Madrid, Spain.

[14] J. Zhao, "Measuring Coupling in Aspect-Oriented Systems", Int.Soft. Metrics Symp. 2004.

[15] S.R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," IEEE Trans. Softw. Eng., vol. SE-20, no. 6, pp. 476–493, June 1994

[16] T.J. McCabe, "A Complexity Measure," IEEE Trans. Softw. Eng., vol. 2, no. 4, pp. 308-320, Dec. 1976.

[17] G. Jay, J. E. Hale, R. K. Smith, D. Hale, N. A. Kraft, and C. Ward, "Cyclomatic Complexity and Lines of Code: Empirical Evidence of a

 ISBN: 978-1-61208-498-5

Stable Linear Relationship," Journal of Software Engineering and Applications, vol. 3, no. 2, pp. 137-143, 2009.

[18] Collaborative Institutional Training (CIIT), 2014, Social & Behavioral Research Modules [Online], [retrieved: June, 2015], Available: https://www.citiprogram.org.

[19] Narayana, Narayana Transaction Anaylser (NTA) Tool [Online], [retrieved: Septemeper, 2015], Available: http://narayana.jboss.org/

[20] SourceForge, Eclipse Metrics Project 1.3.6 [Online]. [retrieved: Jul, 2015] Available: http://metrics.sourceforge.net.

 ISBN: 978-1-61208-498-5