Automatic Documentation of Java Programs

Dujan B. Taha **Asmaa H. Thanoon**

dujan_taha@uomosul.edu.iq

College of Computer Sciences and Mathematics
University of Mosul

Received on: 22/09/2013 **Accepted on: 12/02/2014**

ABSTRACT

Documentation process plays a great role in software systems development and maintenance, and regarded as an important feature for all software projects and programs in general. In the past, documents had been generated manually by documents writers, but these documents were very weak and had many errors, Therefore, automated documentation had been invented due to its good features that is complete, precise, and cheap compared with manual ways.

The work concerns with constructing an automated tool ADT(Auto Documentation Tool) used for documenting source code for programs written in java programming languages in legacy systems particularly and for programs in general which does not have any documentation at all or it were documented poorly in order to make them, and producing as a result of the document process either the class diagrams using UML (Unified Modeling Language) or producing easy understandable textual output using XML (Extensible Markup Language).

Keywords: Automatic Documentation, UML, XML.

التوثيق الالي لبرامج جافا

دجان بشير طه ***اسماء هادي ذنون***

كلية علوم الحاسوب والرياضيات

جامعة الموصل

تاريخ استلام البحث : 2013\9\22 **تاريخ قبول البحث : 2014\2\12**

الملخص

تؤدي عمليات التوثيق دورا كبيرا في تطوير الأنظمة البرمجية وصيانتها، إذ تعد هذه العمليات سمة مهمة لكل مشاريع البرمجيات والبرامج على نحو عام. في البداية كانت الفكرة توليد الوثائق على نحو يدوي من قبل كتاب الوثائق في المجال الخاص بهم وذلك لمعرفتهم بكافة جوانب النظام ولتوليد وثائق أكثر وضوحا، ولكن هذه الطرائق كانت ضعيفة وعرضة للأخطاء, لذلك نشأت محاولات للقيام بعمليات التوثيق على نحو آلي لما يمتاز به التوثيق الالي من مزايا جيدة إذ تكون عمليات متكاملة، دقيقة، وغير مكلفة مقارنة بالطرق اليدوية.

يتعلق البحث ببناء أداة توثيق آلية (ADT(Auto Documentation Tool تستخدم في توثيق شفرة المصدر للبرامج المكتوبة بلغة جافا في الأنظمة القديمة (Legacy systems) ولاسيما البرامج على نحو عام والتي

لا تحتوي على توثيق أو إعادة توثيق هذه الأنظمة بشكل متقن, وقد تم تمثيل ناتج عمليات التوثيق هذه إما على شكل مخطط الصنف (Class diagram) باستخدام لغة النمذجة الموحدة (Unified Modeling Language) UML أو بشكل نصي سهل الفهم باستخدام لغة التوصيف الموسعة (Extensible Markup Language)XML.

الكلمات المفتاحية: التوثيق الالي، XML،UML.

1. المقدمة

إن لعمليات توثيق البرمجيات بمساعدة الحاسوب دوراً فاعلاً في تطوير البرمجيات وهندستها، إذ أنها تساعد مطوري البرمجيات في صيانة هذه البرمجيات في أية مرحلة من مراحل تطويرها وذلك من خلال توفير معلومات قيمة, تشمل المخططات والخوارزميات المستخدمة في تطوير البرمجيات، وتساهم أيضاً في إنتاج برمجيات عالية الجودة من خلال تقليل الأخطاء في كل مرحلة من مراحل تطوير هذه البرمجيات.

لهذا أظهرت عمليات التوثيق الآلي (Automated Documentation) أهميتها في جميع مراحل هندسة البرمجيات وتم الاستفادة من التوثيق الخاص بهذه البرمجيات من قبل مطوري البرمجيات، وبذلك تكون مهمة توثيق البرمجيات مهمة ليست سهلة ولاسيما عند توثيق الشفرة المصدرية (Source Code). أن مطوري البرمجيات يواجهون مهمة صعبة في توثيقها، ويقضون اغلب وقتهم في محاولة فهم الشفرة المصدرية، علاوة على ذلك فان صيانة مثل هذه المشاريع قد يستهلك بشكل إجمالي (70%–90%) من ميزانية دورة حياة مشاريع البرمجيات، وان الحفاظ على جودة التوثيق العالية وتطويرها أمر حيوي لمساعدة المطورين في فهم الشفرة المصدرية وتعديلها [7,6].

سيتم التطرق خلال البحث إلى مفهوم الهندسة العكسية والتي يُعرِفها المختصون في مجال هندسة البرمجيات بأنها العملية التي تأخذ شفرة المصدر كإدخال وتعمل على تحويلها إلى مخطط رسومي يمثل شفرة المصدر في مرحلة التصميم، حيث تعمل الهندسة العكسية على إعادة بناء التقنيات أو المخططات (Charts) الدلالية المفقودة لقواعد البيانات لمراحل تطور مشروع البرمجيات مما يتيح الاحتفاظ بإرث (Legacy) قواعد البيانات بشكل آمن وإن الهدف الأساسي من الهندسة العكسية هو إنتاج أنموذج مفاهيمي لتلك القواعد من خلال شفرة المصدر وبرامج التطبيق [8].

تناولت العديد من البحوث عمليات توثيق البرامج, وهنا سيتم عرض بعض البحوث المختلفة ذات العلاقة بموضوع البحث حيث قام الباحثان ويمر وبيوز [3] في عام 2008 بتطوير أداة آلية بشكل كامل (Fully Automated Tool) وهذه الأداة ساكنة اذ وجد أن التوثيق الناتج من هذه الأداة يكون جيد بنسبة 85% خلال الفترة الزمنية التي تم استخدامه بها, وايضا قام الباحثان كيم ونوتكن [10] في عام 2009 بوصف أداة تسمى LSDiff (Systematic Structural Differences) والتي تعمل على كشف وتوثيق التغييرات التي تحدث على هيكلية الشفرة المصدرية ولكن هنا عملية توثيق الشفرة تكون اكثر تفصيلا, وقامت الباحثة بارون [2] في عام 2010 بدراسة عمليات التوثيق على شفرة المصدر ومن خلال بحثها استطاعت تسهيل الأمر للمستخدمين في كيفية إيجاد التوثيق وكيفية فهمه والتعامل معه وكيفية التحديث عليه, كما وقدم الباحثون دوتافيك و سافت وآخرون [4]

في عام 2011 أداة تعتمد في عملها على العمليات التي تتم على البرمجيات ليتم تحديد عيوب التوثيق وهذا يسهل تحديد جودة عمليات التوثيق.

2. اهداف البحث

يهدف البحث إلى أن تكون أداة التوثيق المقترحة بالمواصفات الآتية:

1- تقوم بتوثيق الشفرة المصدرية بشكل آلي.

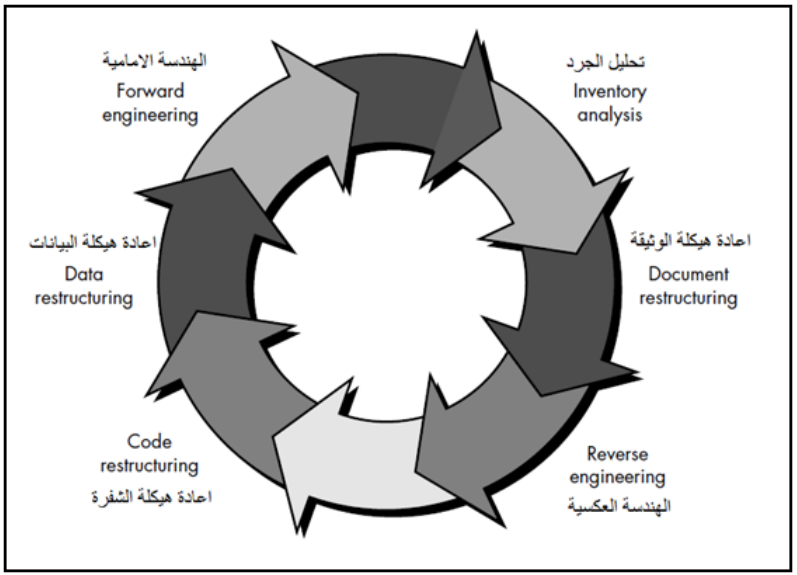
2- تحقيق واحدة من مفاهيم الهندسة العكسية وهي إعادة توثيق البرمجيات (Redocumentation) للبرامج التي تحتوي أصلا على توثيق، فقد يكون هذا التوثيق غير كامل أو غير صحيح أو أن يكون التوثيق دون معنى هذا فضلاً عن توثيق البرمجيات التي لا تحتوي على أي توثيق أو تعليق.

3- تحقيق أسلوب التوثيق المرئي من خلال استخدام مخططات لغة النمذجة الموحدة, فضلا عن تحقيق التكامل ما بين عمل الأداة المقترحة وأدوات هندسة البرمجيات الأخرى من خلال استخدام لغة التوصيف الموسعة في تمثيل البيانات الموثقة.

3. اعادة هندسة البرمجيات والهندسة العكسية

تعرف عمليات إعادة هندسة البرمجيات على أنها دراسة وتعديل أي نظام وإعادة تشكيله بصيغة جديدة أي هي إعادة هندسة وتعديل النظام وعموما تكون هذه العمليات لإغراض إضافة وظيفة جديدة أو لتصحيح الأخطاء. وغالبا ما يطلق على هذه العمليات بالهندسة العكسية وهذا مفهوم خطأ لان عمليات الهندسة العكسية هي جزء من عمليات إعادة هندسة هذه البرمجيات. في الغالب يتم استخدام مفهوم إعادة هندسة البرمجيات في إطار الأنظمة القديمة لان مثل هذه الأنظمة كثيرا ما تحتاج إلى إعادة صياغة أو إعادة تصميم حتى تعمل في بيئات عمل جديدة [13].

ولأن عمليات إعادة هندسة البرمجيات هي عملية إعادة بناء فإنها تحتاج إلى مجموعة من المبادئ التي يجب إتباعها, ومن هذه المبادئ تحديد متطلبات إعادة البناء وهل هذه المتطلبات من الممكن إعادة نمذجتها, وبعد البدء بالعمليات يجب فهم ما هو بالضبط الجزء المطلوب إعادة بنائه, فضلاً عن استخدام تقنيات حديثة في عمليات بناء النظام الجديد حتى ولو كانت مكلفة ولكن ذلك سوف يجنبنا إضاعة أو استهلاك الوقت خلال مرحلة الصيانة وأخيرا تحقيق الجودة في المنتج أو النظام الناتج من خلال عمليات البناء هذه. ولتنفيذ هذه المبادئ يجب استخدام نموذج عمليات إعادة هندسة البرمجيات (Software Reengineering process model) الذي يتضمن ست فعاليات كما موضح بالشكل (1) إذ انه في بعض الأحيان يتم تنفيذ هذه العمليات على نحو متسلسل ولكن هذا لا يحدث



دائما [13].

الشكل (1) نموذج عمليات إعادة هندسة البرمجيات [13]

وفقاً لمعجم (IEEE) تعرَف الهندسة العكسية على أنها عملية استخلاص المعلومات من شفرة المصدر، تعرَف أيضا على أنها ذلك الجزء من البرمجيات الذي يتمثل في استعادة أو إعادة البناء الوظيفي لمواصفاتها التقنية [1]، ويتم ذلك من خلال عملية التعرف على مكونات النظام والعلاقات التي تربط تلك المكونات وتكوين تمثيل للنظام في شكل أخر، كما تعد الهندسة العكسية من أهم فروع التصميم الهندسي وتصنيع التطبيقات خطوة مهمة في دورة تطوير المنتج واستخدامها يؤدي إلى انخفاض كبير في الوقت والتكاليف للمنتج [5].

4. اسباب تنفيذ الهندسة العكسية

هنالك نوعان من المبررات الأساسية لتنفيذ الهندسة العكسية وكالاتي:

أولا: يتم تنفيذ الهندسة العكسية من أجل فهم كيف يمكن لبرنامج الحاسوب أن يعمل فعلاً.

ثانياً: يتم تنفيذ ذلك من اجل تمكين المبرمج من فهم أسباب عدم عمل البرنامج على نحو صحيح.

في الممارسة العملية فإن تطبيق الهندسة العكسية يتم من اجل مجموعة من النقاط من أهمها [11]:

1- وضع برنامج جديد يتعامل مع البرنامج الأصلي.
2- وضع برنامج جديد يتعامل مع بيانات البرنامج الأصلي.
3- إنشاء الوثائق التي تمكن المبرمجين من فهم أفضل للبرنامج الذي تمت عليه الهندسة العكسية.
4- اختبار أمنيَة البرنامج والتحقيق في احتمال خرق الأمنيَة.

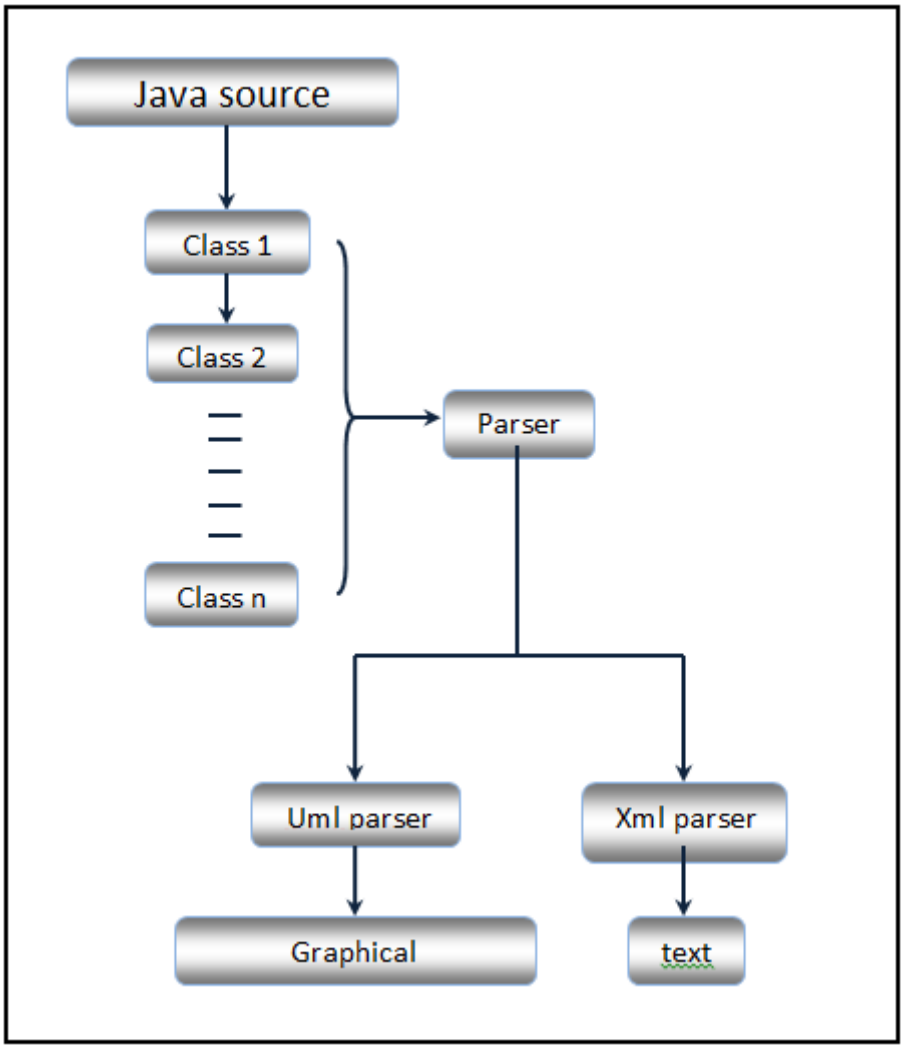
5. دواعي عملية التوثيق

التوثيق مهم في كل مراحل دورة حياة البرمجيات كما هو معلوم ولاسيما مرحلة توثيق شفرة المصدر, حيث يوفر اتصالاً أفضل مابين المتطلبات, يفيد في استخدام التصاميم القديمة, يسهل عملية التكامل ما بين وحدات المشروع المكتوبة على نحو منفصل, وأيضا يفيد في مراجعة التصاميم, كما يزيد من فعالية استخدام الشفرة المصدرية من خلال اعادة استخدامها ويزيد من كفاءة عمليات الاختبار والتحسينات التي يتم اجراءها على المنتج [12]. فضلاً عن انه يجعل أعضاء الفريق في تواصل مستمر, يعمل كمستودع معلومات سيستفيد منه مهندسو الصيانة, يوفر معلومات كافية تسمح لأعضاء ادارة المشروع بإدارة كافة النشاطات كالميزانية والخطط وجدولة عمليات المشروع ويوصف التوثيق للمستخدم كيف يتم تشغيل النظام وإدارته. وأخيرا يجب أن تستمر عملية التوثيق حتى بعد كتابة الشفرة [14,9] .

6. الخوارزمية العامة للأداة المقترحة

تمر الأداة المقترحة ADT بعدد من المراحل والتي تم توضيحها من خلال الشكل (2). إذ تبدأ عملية التحليل بعد اختيار مشروع البرمجيات المراد تحليله والمكتوب بلغة (Java). يتكون المشروع من عدد من الأصناف التي تتشارك فيما بينها لتؤدي المتطلبات الوظيفية للبرمجيات، ومن ثم يبدأ المفسر بعمليات استخلاص المعلومات والعلاقات بين هذه الأصناف لكي يتم بعد ذلك استخدامها من قبل مفسري مخطط الصنف والنص، اذ ان عمل المفسران بشكل متواز يكون من خلال ان المعلومات المستحصلة من تحليل كل صنف بالاضافة الى العلاقات مابين هذه الاصناف يتم استخدامها في نفس الوقت في عملية التوثيق بشكل رسومي مرئي باستخدام مخطط

الصنف وايضا التوثيق بشكل نصي على شكل تقارير XML وهذا يفيد في تقليل الوقت إذ أن كليهما يعتمد على نفس المعلومات والعلاقات التي تم استخلاصها من المشروع البرمجي المراد تحليله لغرض توليد الوثائق.



الشكل (2) المخطط العام للأداة المقترحة

1.6. خوارزمية رسم مخطط UML

تم تمثيل نتائج تنفيذ هذه الخوارزمية من خلال استخدام احد مخططات لغة UML لان ناتج التنفيذ هنا يكون بشكل رسومي مرئي ولغة UML هي واحدة من لغات النمذجة الرسومية التي تقدم لنا صيغة لوصف عناصر او مكونات النظام وذلك من خلال تزويد المستخدمين بلغة نمذجة بصرية ومرئية وهذا يكون اقرب للفهم.

الخطوة الأولى: اختيار احد المشاريع المكتوبة بلغة (Java) لقراءة الشفرة المصدرية الخاصة بها وذلك لغرض توليد توثيق آلي لها.

الخطوة الثانية: تكوين العقدة (node) الخاصة بالمشروع الذي تم اختياره.

الخطوة الثالثة: لكل صنف موجود ضمن المشروع، تحديد الملفات التي تحمل امتداد (.Java).

الخطوة الرابعة: استدعاء المحلل لتحليل محتويات الصنف وتحديد الأجزاء الأتية:

1- تحديد اسم الحزمة.

2- تحديد اسم الصنف.

3- تحديد الصنف الذي يمثل علاقة الوراثة.

4- تحديد الصنف الذي يمثل علاقة تمثيل الواجهات.

5- تحديد الدوال فضلا عن دوال البناء.

6- تحديد الأنواع التي تم تعريفها ضمن كل صنف.

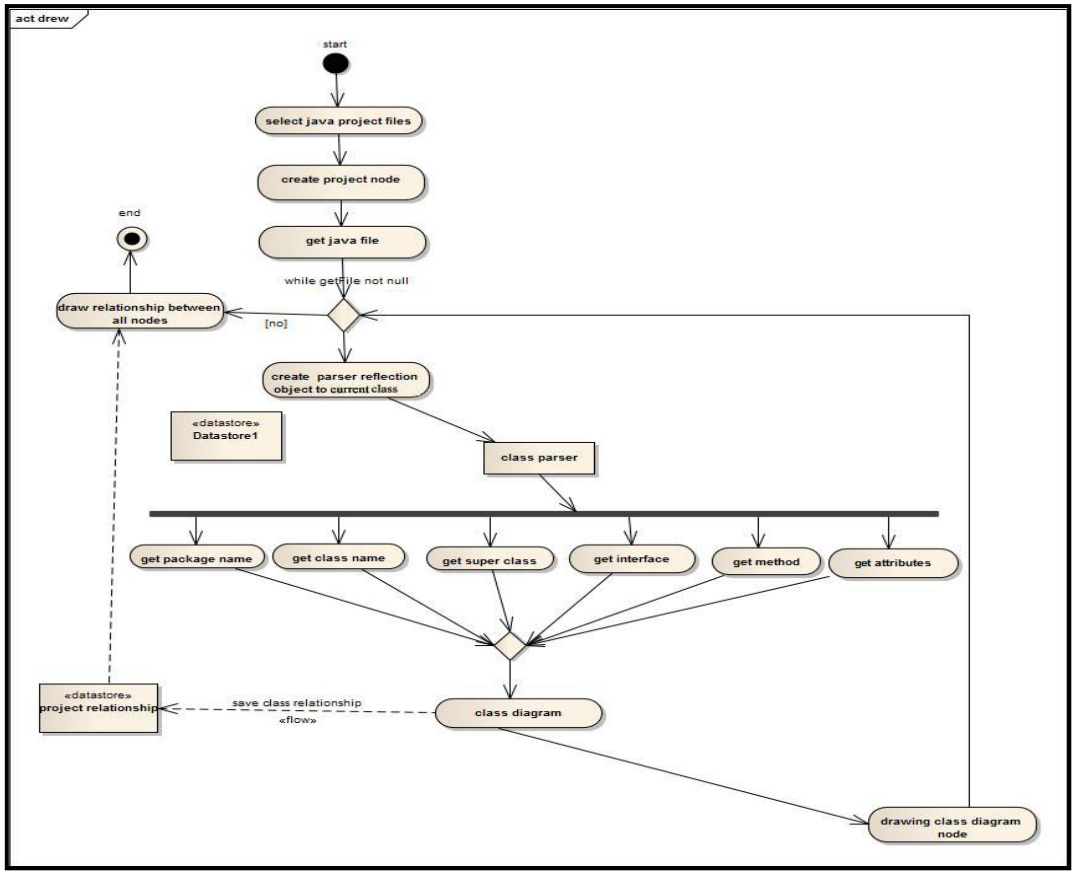
وبعد تحديد الأجزاء أعلاه ضمن الصنف الواحد يتم استدعاء بقية الأصناف تباعا إلى حين انتهاء جميع الأصناف المكونة للمشروع المراد توثيقه، وبعدها يتم خزن المعلومات المحددة.

الخطوة الخامسة: خزن نوع العلاقة مابين أصناف المشروع والتي إما أن تكون علاقة وراثة أو علاقة تمثيل الواجهات وأحيانا الاثنين معا.

الخطوة السادسة: رسم العقدة الخاصة بكل صنف ضمن مخطط الصنف مع توضيح محتويات كل صنف وما يحتويه من أجزاء.

الخطوة السابعة: تمثيل العلاقات مابين العقد الموجودة ضمن مخطط الصنف والتي تمثل الأصناف المكونة للمشروع وتوضيح هذه العلاقة بالرسم الذي يظهر على واجهة المستخدم الرسومية.

و قد تم توضيح خطوات خوارزمية رسم مخطط UML من خلال استخدام مخطط الأنشطة (UML activity diagram) الموضح بالشكل (3).



الشكل (3) مخطط الأنشطة لخوارزمية رسم مخطط UML

1.6. خوارزمية توليد توثيق تقارير XML

الخطوة الأولى: اختيار احد المشاريع المكتوبة بلغة (Java) لقراءة الشفرة المصدرية الخاصة بها.

الخطوة الثانية: تكوين العقدة الخاصة بالمشروع الذي تم اختياره.

الخطوة الثالثة: لكل صنف موجود ضمن المشروع، تحديد الملفات التي تحمل امتداد (.Java).

الخطوة الرابعة: استدعاء المحلل وإتباع مايأتي:

1- تحديد العناصر ضمن كل صنف.

2- إيجاد العلاقات ما بين الأصناف.

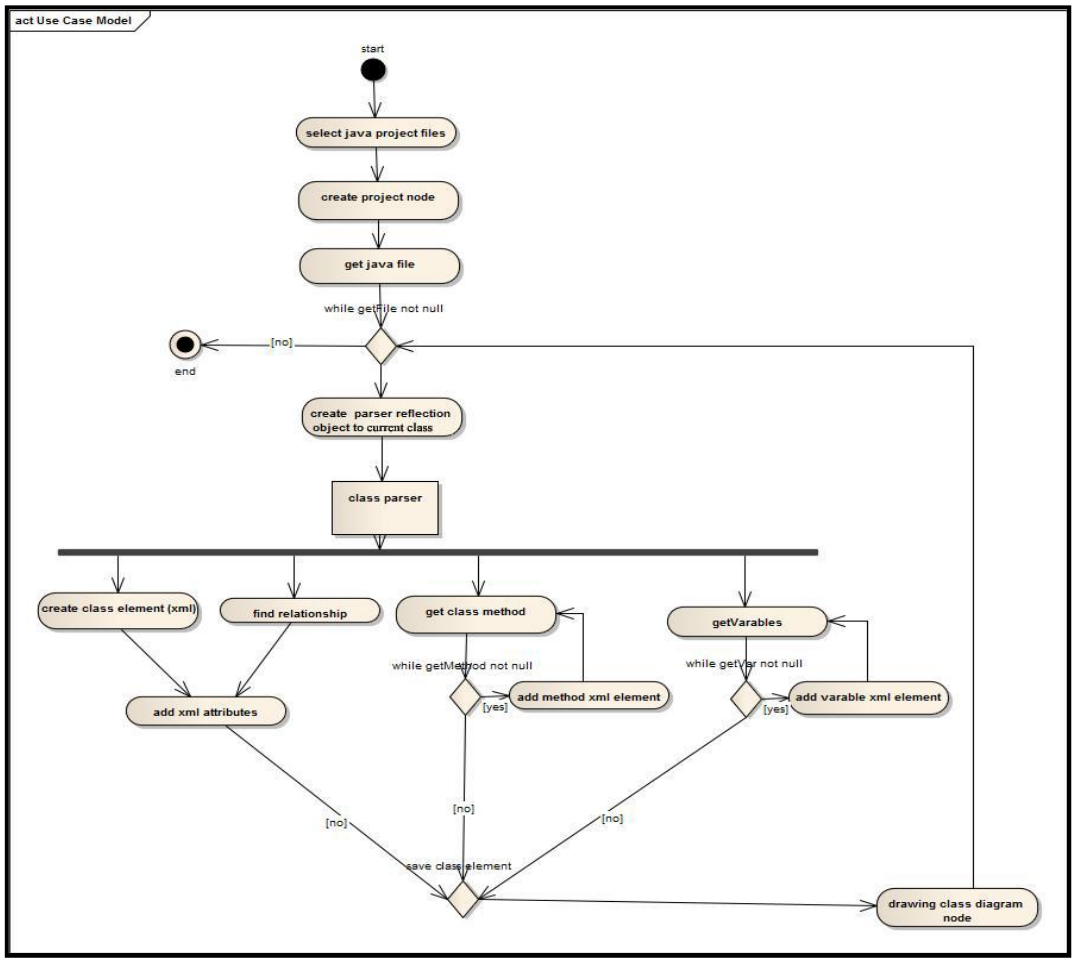
للخطوتين (1) و (2) إضافة المعلومات الناتجة من عملية التحليل وخزنها كصفات (Attributes) في ملف XML.

3- لكل صنف ضمن المشروع تحديد جميع الدوال الموجودة في الصنف الواحد وإضافتها كعناصر إلى ملف XML.

4- تحديد المتغيرات داخل كل صنف وإضافتها إلى ملف XML.

الخطوة الخامسة: تعاد الخطوات (1)، (2)، (3) و (4) إلى حين انتهاء تحليل جميع الأصناف المكونة للمشروع.

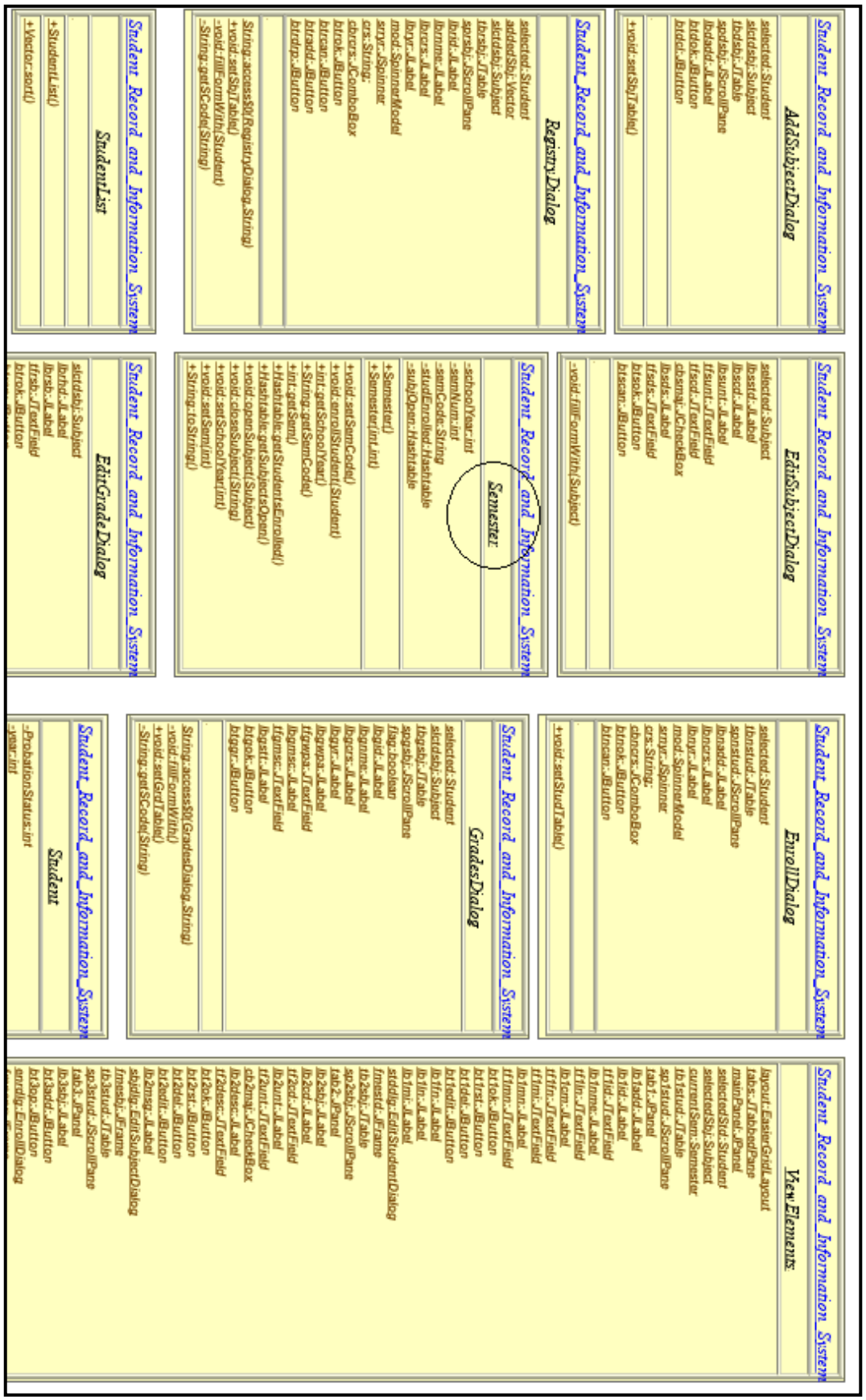
وقد تم توضيح خطوات خوارزمية توليد تقارير XML من خلال إستخدام مخطط الأنشطة الموضح بالشكل (4).



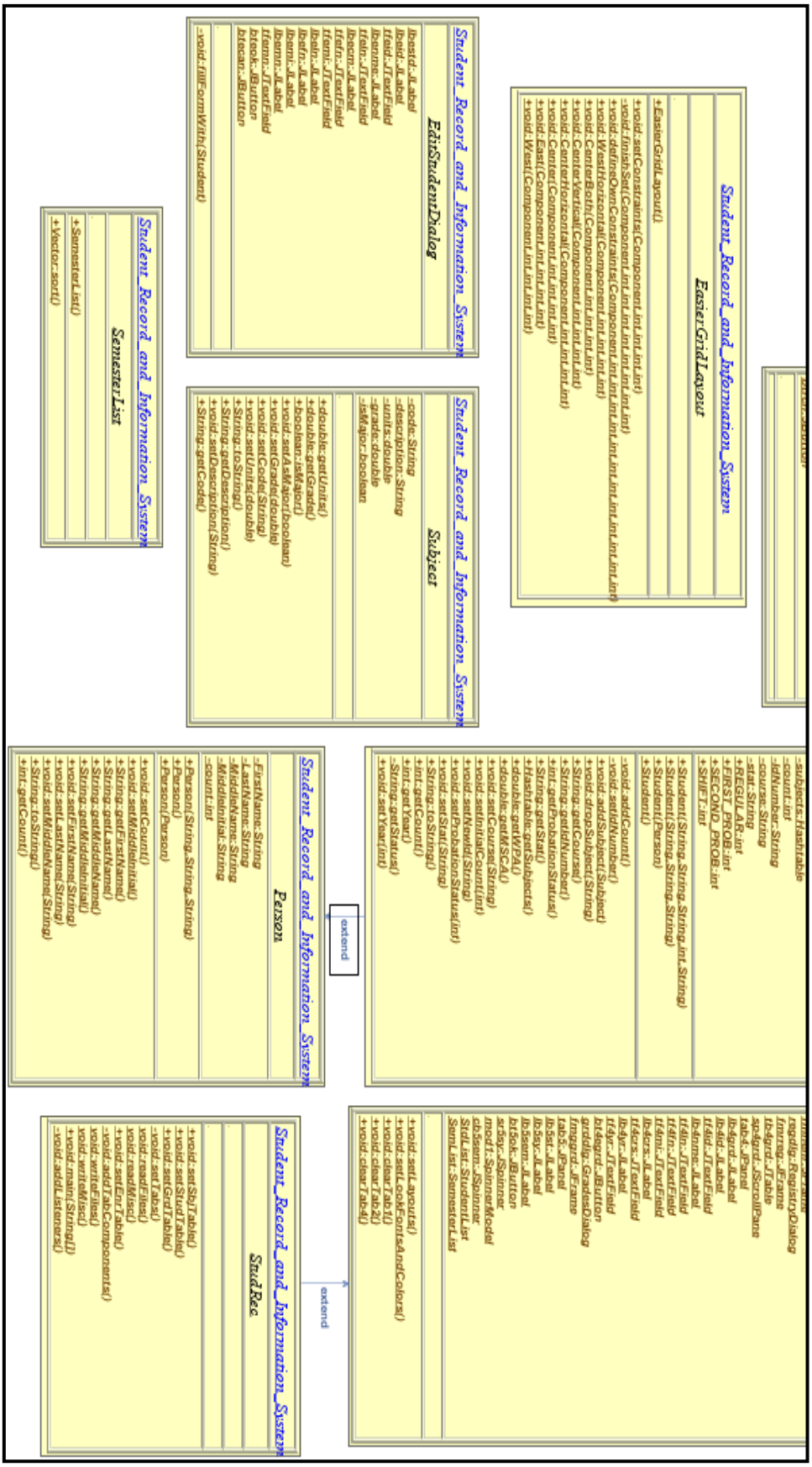
الشكل (4) مخطط الأنشطة لخوارزمية توليد تقرير XML

7. النتائج

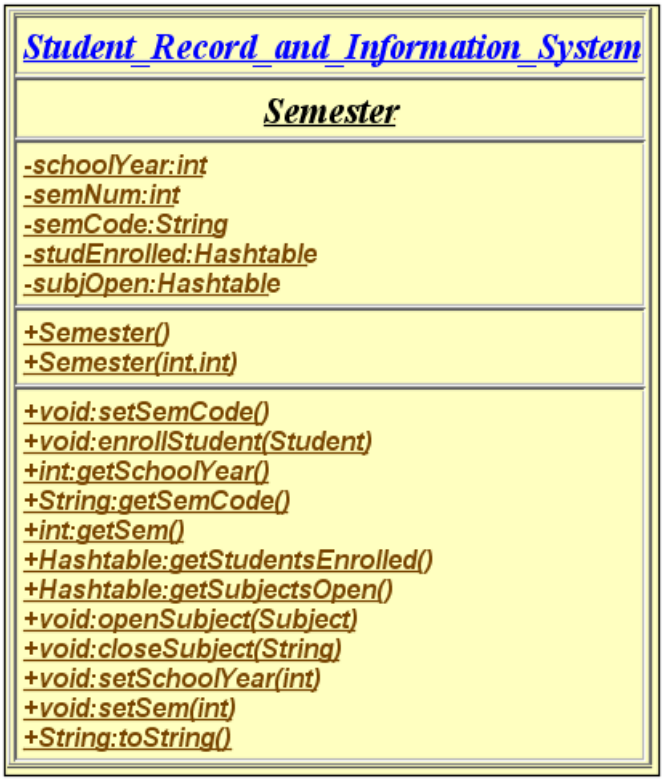
تم تطبيق الأداة المقترحة على أكثر من نظام برمجي وذلك لتوضيح قدرة النظام على تمثيل علاقات الوراثة وتمثيل الواجهات فضلا عن إمكانية الأداة على توثيق المكونات الأخرى للشفرة المصدرية, وأحد هذه الانظمة هو نظام الطلاب حيث يتكون هذا النظام من ستة عشر صنفاً, اثنين منها تحوي علاقة وراثة, والشكل (5– أ) يمثل مخطط الصنف الخاص بهذا النظام ككل, وقد تم تحديد الصنف (Semester) بالشكل الدائري لعرض مكوناته بشكل تفصيلي من خلال الشكل (5– ب) اما علاقة الوراثة فقد تم تحديدها بالشكل المستطيل وكما موضح بالشكل (5– أ).



الشكل (5- أ) مخطط الصنف الخاص بتمثيل نظام الطلاب



الشكل (5- أ) مخطط الصنف الخاص بتمثيل نظام الطلاب(تكملة)



الشكل (5– ب) مخطط الصنف الخاص بتمثيل الصنف (Semester)

بعد تطبيق الاداة ADT على نظام الطلاب نلاحظ انه تم توثيق كافة الاصناف المكونة للنظام بضمنها الصنف (Semester) وأيضا تم توثيق كافة الانواع والدوال ودوال البناء المكونة لكل صنف فضلا عن تحديد وتوثيق علاقة الوراثة وهذا موضح بالشكل (5– أ) و (5– ب), وعملية التوثيق هنا كانت بشكل رسومي مرئي.

والشكل (6) يوضح توثيق الصنف (Semester) بشكل نصي (تقرير XML), اذ نلاحظ من الشكل انه في البداية تم تحديد اسم الصنف ومن ثم تحديد توثيق علاقة الوراثة من خلال تحديد اسم الصنف المتوارث, وبعدها تحديد اسم الصنف الذي يمثل علاقة الواجهات في حال وجودها, فضلا عن تحديد الانواع المعرفة وبعد ذلك تم تحديد دوال البناء ومن تعريف الدوال من خلال تحديد اسمها ونوعها هل هي عامة ام خاصة ام محمية وتحديد نوع واسم المتغيرات المستلمة مع تحديد نوع الارجاع ومن ثم انهاء الصنف بعد تحديد مكوناته, وبنفس الطريقة يتم توثيق الاصناف الاخرى الخاصة بالنظام.

الشكل (6) توثيق الصنف (Student) باستخدام لغة XML

8. الاستنتاجات والاعمال المستقبلية

تم في هذا البحث تصميم وبناء اداة الية ADT لتوثيق البرامج وتميزت الاداة عن الاعمال السابقة بما يلي:

- تم تطبيق الاداة على البرامج القديمة التي لا تحتوي على توثيق اصلا او الانظمة التي تحتوي على توثيق ولكنه غير متكامل.
- الاداة لها القابلية على توليد توثيق في كل عملية تنفيذ للنظام على شفرة المصدر وبعد إجراء أي تحديث أو إضافة عليه فضلا عن ان استخدام الأداة المقترحة سوف يقلل من الجهد والوقت وكلا العاملين مهم في مشاريع هندسة البرمجيات.
- إثبات أهمية لغة التوصيفات الموسعة في بناء الوثائق من خلال توليد توثيق متكامل خالي من التعارضات والتكرارات.
- تنفيذ واحد من أهداف الهندسة العكسية من خلال تطبيق عمليات إعادة التوثيق على الشفرات البرمجية والبيانات التي تتعامل معها هذه الشفرات، وذلك من خلال إتباع أسلوب منهجي ومنظم (أداة آلية).
- توثيق كافة الأنواع المعرفة والدوال ودوال البناء وعلاقات الوراثة وتمثيل الواجهات بالإضافة إلى دقة نتائج التوثيق من خلال مقارنتها مع شفرة المصدر الأصلية للمشروع المعني بعملية التحليل.
- التوثيق الناتج مقروءً (Readable) وواضحاً وهذا يسهل فهم الشفرة بشكل سريع وأيضا سهولة تتبع التغييرات ويكون تحقيق ذلك من خلال استخدام المخططات والأشكال الرسومية التي تكون اقرب للفهم.

وفيما يخص الأداة المقترحة فان من أهم التوصيات المستقبلية لتطويرها هي عمل تكامل لإخراج الأداة المقترحة مع عمل أداة أخرى من ضمن أدوات هندسة البرمجيات, وايضا تنفيذ عمليات الهندسة العكسية لتحويل الشفرات المكتوبة بلغات البرمجة الكيانية الموجهة (Object-oriented programming) OOP إلى نماذج أخرى مكتوبة بلغة النمذجة الموحدة مثل مخطط الانشطة أو أي أنموذج آخر.

المصادر

[1] Abraham Silberschatz, Henry F. Korth and S. Sudarshan, (2006), "Database System Concepts", McGraw-Hill.

[2] Baron, R., (2010), " Discrimination in the Documentation of Open Source Software ", Degree of Bachelor of Science, Professional Writing, Worcester Polytechnic Institute, USA.

[3] Buse, Raymond P.L., Weimer ,Westley R., (2008), " Automatic Documentation inference for exceptions", ISSTA '08 Proceedings of the 2008 international symposium on Software testing and analysis, PP: 273- 282, ISBN: 978-1-60558-050-0, doi>10.1145/1390630.1390664.

[4] Dautovic A., Plosch R. , Saft M., (2011), "Automatic Checking of Quality Best Practices in Software Development Documents", Quality Software (QSIC), 11th International Conference, IEEE Computer Society, PP: 208-217.

[5] Eyup Bagci, (2009), "Reverse engineering applications for recovery of broken or worn parts and re-manufacturing: Three case studies", Advances in Engineering Software(Elsevier), Vol.40 p.p 407–418.

[6] Friesen, A., Lemcke, J., Oberle, D., Rahmani, T., (2008), "Description of Functional and Non-Functional Requirements", SAP Research CEC, Karlsruhe, Germany, Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Seventh Framework Programme, http://most-project.eu.

[7] Hallam P.,(Jan 2006), "What do programmers really do anyway?", In Microsoft Developer Network (MSDN) | C# Compiler.

[8] Jean-Luc Hainaut, (2009), "Legacy and Future of Data Revers Engineering",16th Working Conference on Reverse Engineering, IEEE p.p.4-4.

[9] Kamthan, P., (2007), "On the Prospects and Concerns of Integrating Open Source Software Environment in Software Engineering Education", Concordia University, Montreal, Quebec, Canada, Journal of Information Technology Education, Volume 6.

[10] Kim M. and Notkin D., (2009), "Discovering and representing systematic Code changes", In International Conference on Software Engineering, pages 309-319.

[11] Leif Gamertsfelder, (2003), "Software reverse engineering – the current state of Australian law", Computer Law & Security Report (Elsevier), Vol.19 no. 5 p.p 394-400.

[12] Parnas, D. L., (2011), "Precise Documentation: The Key to Better Software", The Future of Software Engineering, Springer Berlin Heidelberg, pp :125-148, DOI:10.1007/978-3-642-15187-3_8.

[13] Pressman, R., (2010), "Software Engineering: A Practitioner's Approach", 7th Edition, McGraw-Hill, New York, USA, ISBN 978-

0-07-337597-7.

[14] Sommerville, I., (2001), "Software documentation", Literate progamming. Lancaster University, UK, Retrieved from www.literateprogramming.com/documentation.pdf.