A Bisimulation-based Hierarchical Framework for Software Development Models

Ping Liang
College of Computer Science Technology, Southwest University for Nationalities, Chengdu, China
Email: liang_p@hotmail.com

Abstract—Software development models have been ripen since the emergence of software engineering, like waterfall model, V-model, spiral model, etc. To ensure the successful implementation of those models, various metrics for software products and development process have been developed along, like CMMI, software metrics, and process re-engineering, etc. The quality of software products and processes can be ensured in consistence as much as possible and the abstract integrity of a software product can be achieved. However, in reality, the maintenance of software products is still high and even higher along with software evolution due to the inconsistence occurred by changes and inherent errors of software products. It is better to build up a robust software product that can sustain changes as many as possible. Therefore, this paper proposes a process algebra based hierarchical framework to extract an abstract equivalent of deliverable at the end of phases of a software product from its software development models. The process algebra equivalent of deliverable is developed hierarchically with the development of the software product, applying bi-simulation to test run the deliverable of phases to guarantee the consistence and integrity of the software development and product in a trivially mathematical way. And an algorithm is also given to carry out the assessment of the phase deliverable in process algebra.

Index Terms—Process Algebra, Bi-Simulation, Software Development Models, Hierarchical Framework, UML, Software Consistence

I. INTRODUCTION

Software development models have been standardized over a decade. It has provided guidance to facilitate enterprises to normalize their development behaviors so as to create more reliable software products. The typical models include waterfall, V-model, and Spiral, etc. The goal of these models is to create a product to last as long as possible in a process of software evolution. There are also other types of software development models that are for software products required to be developed in a short time or for a short term or a specific purpose, like agile development and extreme programming, etc. No matter which software development model is in use, consistent abstract integrity should be vital but hard to be achieved, as in [1], from software engineering’s point of view.

In the software development history, object-oriented approach has been favored since 1990s by developers. UML [18] is its powerful tool to depict a system in the system's visual graph way. The tool can be applied in all the phases along any software development model to facilitate the analysis and understanding of a software product by showing the logical process of the software. The tool includes mainly 9 diagrams. Each diagram depicts a system from its point of view and all of them give a whole picture. The implementation of UML to better software development motivated the software architecture based on the technology, e.g. 4+1 view, etc. [16] [17] [21] The integrity required by software engineering in a software architecture [15] [18] demands the correspondence between UML diagrams. The abstract integrity guarantees the uniform development of a software system. The integral of the system makes it reliable and available. However, as a graph tool, its lack of accuracy is inevitable even UML has its syntax well defined. The phased software development is like rallying messages from analysis to design and implementation or passing messages via communication system, and some distortion is bound to happen. With the loose and abstract depiction of UML, the original business intention can hardly be fully re-implemented by its software substitute.

Many developers and scholars have worked hard to amend the shortcomings. The application of software development models, e.g. spiral model, agile model and extreme model, etc., helps the consistence and integrity of developed systems. There still lacks an efficient way to eliminate the inconsistence due to the complexity of software products. Learning from communication systems, this paper is to borrow the idea of process algebra to provide the accuracy that text and graph modeling cannot offer for their inherited nature. In terms of accuracy, process algebra achieves it by its mathematic description and logic inference [2]. Unfortunately, the good intent cannot be realized because the real world cannot be perfect in a mathematical or logic way. To address this issue, this paper proposes using UML models to extract a holistic picture of a software product in process algebra which serves as a reference for development and maintenance of a software product. The idea is as follow,

Obtain a detailed and complete set of documents of a software product. Then to draw a corresponding relation map to elaborate the coalition and association between diagrams viewing a system from different angles. Using UML rather than text is to simplify the process abstraction.
To put associated symbols from diagrams into a set of tokens. Each symbol becomes an individual token. A process in communication systems consists of control flow and data. And a business process includes resources as well. Symbols from UML diagrams represent data, control flow and resource. A perfect set can be made for covering information needed by a business process.

From a set of tokens to a process in process algebra, the first is to use activity diagram as main stream because a process is composed by atomic processes. The activity in an activity diagram is similar to a process activity. The classes from class diagram are logical relations with the definition of data structure. And sequence and collaboration diagram specify the sequential and associated behaviors of activities further. The state-chart can provide the state the system is in to describe the outcome of an activity in a process.

And based in the figure above, the data structure of process algebra of a system from the system's UML can be generated, so can its equation and proof. The reasons are,

- The process algebra can clarify an activity in the way of white box. As in structural operational semantics, not only input and output are required to be same, but also the branching. The process algebra notation of a software development phase can compare with one of its former and latter phase by bi-simulation. Inconsistence shall be discovered and dealt with. The benefit is that the relevant consistence and integrity can be achieved.

The paper is arranged like this. The first section gives introduction to issues in software development and a concept of our solution; the second section is to elaborate how to associate UML diagrams; the third section is to generate a data structure of a system by process algebra and the forth section to prove the equivalence of process algebra represented phases by bi-simulation; and the fifth section is to...
section is to summarize the advantages and disadvantages of the solution proposed by this paper.

II. CORRESPONDENCE BETWEEN UML DIAGRAMS

The reason of choosing UML diagrams rather than the text of requirement is due to the following [18],

- Natural text is far too complicated to get its semantics extracted;
- UML diagrams are supported by standardized syntax, which provides relatively trivial but comprehensive definition for the logical view of its corresponding requirement in natural text, in a way to be an excellent substitute for natural text;
- UML is a visual graphic language which models a software product logically. Its logics is equivalent to that of mathematics and the development of a software product is reckoned as a process for decades. So it is possible to map UML diagrams to the notation of process algebra.

Based on the preceding reasons, we give a correspondence map of use case, class, sequence and activity diagram in Fig. 1.

As shown in the figure, the activity program is the closest representation of a process [4] but not a complete match. Other diagrams can supplement the activity diagram. The class diagram provides classes participating in activities and the sequence diagram clarifies the order of activities. Use case diagram overviews the system in a general sense. The granularity of activity diagram and sequence diagram determines the behavior of a software product. The state-charts give the status of the system which is in activities as below in Fig. 2.

III. PROCESS ALGEBRA

Process algebra was developed for communication systems to describe the concurrency [5] [9]. So it is a structural operation behavior language which can depict the control flow, data flow and resource allocation. The process algebra consists of the set for activities with the system status attributed by data flow and resource flow. So to map UML diagrams to process algebra, let us set the process algebra model to be \( T(A) \), with \( a_0, ..., a_i \), \( s_0, ..., s_i \), and \( d_0, ..., d_i \in A \), or a pair \((s, P)\) or \((d, P)\) \( \in A \). \( a_0, ..., a_i \) are activities, \( s_0, ..., s_i \), are states of a software product and \( d_0, ..., d_i \) are data structure. So we can interpret the model as \( A \) referring to a set of activities specified in activity diagram. An individual activity is further attributed by its corresponding states of the involved data structure. Therefore, based on the deduction rules of process algebra [2], the expression of activity diagram in process algebra would be the sequential activities,

\[
a_1 \bullet a_2 \bullet a_3 \bullet a_4 \bullet a_5
\]

and the deduction would be,

\[
(s, d, r_1) \xrightarrow{a_1} (s_2, d_2, r_2) \xrightarrow{a_2} (s_3, d_3, r_3) \xrightarrow{a_3} (s_4, d_4, r_4) \xrightarrow{a_4} (s_5, d_5, r_5)
\]

among which \( s, d, r \) represents state, data, and resource respectively.

IV. BI-SIMULATION EQUIVALENCE

Before any actual works start, there are several issues must be addressed. The first is that the granularity of UML diagrams must match up with process algebra so that the behavior of a software product can be clearly defined. As process algebra is originally is to define processes of hard-coded systems and structural operation behavior language, the granularity shall be tuned to suit the abstract level of each phase. So a hierarchical framework is needed [11] [12]. The second is that the logic of UML diagrams works as the semantics conforming to the syntax of process algebra. However, UML diagrams can only be trivially proved by its process algebra model in the following two ways,

- The modeling of a software product is an ongoing process in software development. The granularity of an early phase is inevitably coarser than that of a latter phase. In our hierarchical framework, it is only realistic to ensure the consistence of phases in the granularity of an earlier phase. So it is only able to prove the consistence of phases trivially;
- The hierarchical framework proposed by this paper can only ensure the consistence of phases to facilitate achieving quality of a software product in order to make a software product more reliable and retractable. It does not address the issue of user satisfaction. So the quality of a software product also relies on software metrics and can be ensured by our framework trivially.

Nonetheless, our framework can be helpful significantly, even trivially. So in order to enforce the consistence of software development phases, it is required the deliverable of next phase is in conformance with that of last phase in the earlier phase’s granularity so as to keep the system integrity. Our proposal is to prove the consistence by bi-simulation of the phase models by process algebra.

Requirement analysis is inherently abstract. Design is more detailed and implementation of coding shall be more specific. After finishing the detailed phase later in the development process, the first is to prove that the system model of the latter phase is bi-simulation to that of the former phase in the same granularity and then rebuild the former phase in a lower hierarchical level based on that of the latter phase [7] [9].

As shown in Fig. 3, the requirement \( a_1 \) is specified as \((a_11, a_12), a_13)\) in design and the \( a_{11} \) in design becomes \( a_{111}, a_{112}, a_{113}\) in implementation. The \( a_i \) is in a coarser granularity than \( a_{i1}, a_{i2} \) and \( a_{i3}\), and \( a_{111}, a_{112}, \) and \( a_{113}\) are further detailed. To demonstrate the trivial consistence proposed earlier in this paper, the proof process is to use
the intermediate phase – design phase to show how the process algebra model in the beginning of a phase can firstly compare to the model in the former phase and then be elaborated to become the model as the delivery for the next phase. So in the design phase, there shall have two process models. One is created at the beginning of the design phase in the same granularity as the requirement process models. One is created at the beginning of the design phase and the other is in the same granularity as the requirement process model. Therefore, based on the theorems in [2], we need to prove,

\[ T_{\text{req}}(A_{\text{req}}) \equiv T_{\text{des}}(A_{\text{des}}) \]

and,

\[ T_{\text{des}} \equiv T'_{\text{des}} \]

Then,

\[ T_{\text{des}}(A'_{\text{des}}) \equiv T_{\text{imp}}(A_{\text{imp}}) \]

In the expression above, \( T_{\text{req}}(A_{\text{req}}) \) refers to the process algebra model of requirement phase, \( T_{\text{des}}(A_{\text{des}}) \) for design phase in the granularity of requirement and \( T'_{\text{des}}(A'_{\text{des}}) \) in the finer granularity; \( T_{\text{imp}}(A_{\text{imp}}) \) for implementation phase. And \( T_{\text{des}}(A_{\text{des}}) \) and \( T'_{\text{des}}(A'_{\text{des}}) \) are the process algebra model in design phase in different granularity. Due to the different granularity of the process algebra model at the beginning and end of the design phase, the two models of the design phase can only be proved consistent trivially.

To prove the equivalence, we firstly start with a general process to demonstrate how activity in the former model can be mapped into the model in the next stage. Let us start with a sequence of activities,

\[ \alpha_1 \alpha_2 \cdots \alpha_i \alpha_i \cdots \alpha_n \]

\[ \Rightarrow \]

\[ (\alpha_1 + \cdots + \alpha_n) \alpha_i \alpha_i \cdots \alpha_n \cdots \alpha_n \]

In the formula above, each activity \( \alpha_i \) in the former phase is able to be divided into \( (\alpha_1 + \cdots + \alpha_n) \) in the later phase. So the process algebra model can be shown in different granularity. Here we only showcase the replacement of \( \alpha_1 \). The rest of the formula would be the same.

Now as specified in the last section, the activities cause the change of the model’s state, data and resource. So the set of the system’s state, data structure and resource changes from one to another by executing the activity \( \alpha_i \)

\[ (s_1, d_1, r_1) \xrightarrow{\alpha_i} (s_2, d_2, r_2) \]

Then the formula above means that the system can transfer from \((s_1, d_1, r_1)\) to \((s_2, d_2, r_2)\) via the activity \( \alpha_i \) or the sequence of its sub-activities \( \alpha_{i1} \cdots \alpha_{in} \). Therefore,

\[ \alpha_i \equiv \alpha_{i1} \cdots \alpha_{in} \]

So we show that the activities in the former phase can be proved to be in consistence with those in the finer granularity in the latter phase. This generalized deduction can be used for further proof. So the following equations can be obtained,

\[ \{a_{\text{reqi}}\} \equiv \{a_{\text{degi}}\} \]

\[ \{a_{\text{desi}}\} \equiv \{a_{\text{impi}}\} \]

\[ \Rightarrow \]

\[ T_{\text{req}}(A_{\text{req}}) \equiv T_{\text{des}}(A_{\text{des}}) \]

\[ T(A_{\text{des}}) \equiv T' \]

\[ T'_{\text{des}}(A'_{\text{des}}) \equiv T_{\text{imp}}(A_{\text{imp}}) \]

As seen in the equation above, the relation between \( T_{\text{des}}(A_{\text{des}}) \) and \( T'_{\text{des}}(A'_{\text{des}}) \) adopts the symbol of \( \equiv \) rather than \( \approx \). The reason is that \( T_{\text{des}}(A_{\text{des}}) \) and \( T'_{\text{des}}(A'_{\text{des}}) \) are the model in different granularity. It is possible to prove the equivalence of definition of the two models but not meaningful to prove the branching bi-simulation equivalence between them due to the non-silent transitions existing in \( T'_{\text{des}}(A'_{\text{des}}) \). [13]

With the deduction of process algebra, we can define an algorithm for the verification and comparison of process algebra models as below.

1. Read \( T_{\text{req}}(A_{\text{req}}) \);
2. Read \( T_{\text{des}}(A_{\text{des}}) \);
3. While \((a_{\text{reqi}} \rightarrow \emptyset)\):
   4. Compare the pair \((s_{\text{reqi}} \mid d_{\text{reqi}} \mid r_{\text{reqi}}, \alpha_{\text{reqi}})\) with \((s_{\text{desi}} \mid d_{\text{desi}} \mid r_{\text{desi}}, \alpha_{\text{desi}})\);
   5. If any inconsistence
   6. Then modify the inconsistence;
   7. Read \( T'_{\text{des}}(A'_{\text{des}}) \);
   8. Define the granularity;
   9. While \((a_{\text{degi}} \rightarrow \emptyset)\):
   10. If \((s_{\text{degi}} \mid d_{\text{degi}} \mid r_{\text{degi}}, \alpha_{\text{degi}})\)
   11. Compare \((s_{\text{degi}} \mid d_{\text{degi}} \mid r_{\text{degi}}, \alpha_{\text{degi}})\) with \((s'_{\text{degi}} \mid d'_{\text{degi}} \mid r'_{\text{degi}}, \alpha'_{\text{degi}})\) in same granularity
   12. If any inconsistence
   13. Then modify the inconsistence;
   14. Read \( T'_{\text{imp}}(A) \);
   15. While \((\alpha'_{\text{impi}} \rightarrow \emptyset)\):
   16. Compare \((s'_{\text{impi}} \mid d'_{\text{impi}} \mid r'_{\text{impi}}, \alpha'_{\text{impi}})\) and \((s''_{\text{impi}} \mid d''_{\text{impi}} \mid r''_{\text{impi}}, \alpha''_{\text{impi}})\);
   17. If any inconsistence
   18. Then modify inconsistence;
   19. End

The algorithm given above shows that the algorithm we propose can be automatically processed by computer.
However, there are possible issues that need to be addressed. When mapping the activities in design phase, each activity in the model in the beginning of design is divided into many finer activities as the result of the detailed design. This requires a definition of the correspondence between \( a_{desi} \) and \( a'_{desi} \) to ensure the correct comparison of \( T_{desi}(A_{desi}) \) and \( T'_{desi}(A'_{desi}) \). So the algorithm above is modified as below:

1. Read \( T_{req}(A_{req}) \);
2. Read \( T_{desi}(A_{desi}) \);
3. While! (\( a_{req} \rightarrow \checkmark \))
4. Compare the pair \( (s_{req} | d_{req} | r_{req}, a_{req}) \) with \( (s_{desi} | d_{desi} | r_{desi}, a_{desi}) \);
5. If any inconsistence
6. Then modify the inconsistence;
7. Read \( T'_{desi}(A'_{desi}) \);
8. Define the granularity;
9. Set up the correspondence table between \( (s_{desi} | d_{desi} | r_{desi}, a_{desi}) \) and \( (s'_{desi} | d'_{desi} | r'_{desi}, a'_{desi}) \);
10. Read \( a_{desi} \);
11. j=0;
12. flag=false;
13. while! (\( a'_{desi} \rightarrow \checkmark \))
14. Compare \( (s_{desi} | d_{desi} | r_{desi}, a_{desi}) \) to \( (s'_{desi} | d'_{desi} | r'_{desi}, a'_{desi}) \)
15. If \( (s_{desi} | d_{desi} | r_{desi}, a_{desi}) \) == \( (s'_{desi} | d'_{desi} | r'_{desi}, a'_{desi}) \)
16. flag = true;
17. If flag == false
18. Then modify the inconsistence;
19. Read \( T_{imp}(A) \);
20. While (\( a'_{impi} \rightarrow \checkmark \))
21. Compare \( (s'_{impi} | d'_{impi} | r'_{impi}, a'_{impi}) \) and \( (s_{impi} | d_{impi} | r_{impi}, a_{impi}) \)
22. If any inconsistence
23. Then modify inconsistence;
24. End

It can be seen from the modified algorithm that the comparison of the process algebra models in the same phase address the issue of matching up the activities in coarser granularity with those in the finer granularity. And this algorithm can be applied in any software life cycle models, particularly useful for the waterfall model. The algorithm can be applied directly for the waterfall. As for other software models, the algorithm only need to be given minor changes.

Applying the algorithm for the spiral model, the algorithm can be,
1. While! (Final product delivery)
2. Carry out the original algorithm as above;
3. Update \( T_{req}(A_{req}) \), \( T_{desi}(A_{desi}) \) and \( T_{imp}(A) \)
4. Continue;
For the rapid development model,
1. Create \( T_{req}(A_{req}) \), \( T_{desi}(A_{desi}) \) and \( T_{imp}(A) \) for the prototype model;
2. While! (Final product delivery)
3. Modify \( T_{req}(A_{req}) \), \( T_{desi}(A_{desi}) \) and \( T_{imp}(A) \) to create the model for the software to be developed;
4. Carry out the original algorithm as above;
5. Update \( T_{req}(A_{req}) \), \( T_{desi}(A_{desi}) \) and \( T_{imp}(A) \)
6. Continue;
For Agile development process, there are many methods that can be adopted for tailoring the software development, e.g. XP programming, formal method, or IBM RUP, etc. However, the general methodology of agile development process is the same. The key point to create a “To-Do” list in each iteration of development. So the algorithm can be like below,
1. While! (Final product delivery)
2. Create a “To-Do” list;
3. Create \( T_{req}(A_{req}) \), \( T_{desi}(A_{desi}) \) and \( T_{imp}(A) \) for the “To-Do” list;
4. Carry out the original algorithm as above;
5. Continue;
As for model-driven engineering, this development process invented by OMG and based on abstract modeling of software systems is extremely suitable for the application of the method proposed by this paper. The activity diagram can be certainly the part of the development documents in all development phases. It is convenient to generate a process algebra model for the software system in each phase so as to compare the consistence. And other development methods, like SOA or Aspect-oriented development, can also be modeled and the method be applied.

As shown above, modeling of software systems from activity diagram to generate its process algebra model and then to use the algorithm above to ensure the consistence in the life-cycle of the software system can be applicable for various software systems.

V. CONCLUSION

The preceding discussion can conclude that to model a software development process in process algebra in compliance with UML diagrams is possible to strengthen the link between phases and enforce the consistence between phases. It is an approach to maintain the abstract integrity necessary for a robust software product to cope with changes in software evolution. Due to the logical deduction that can be carried out through the process algebra model, there is even the possibility of automating the comparison between the models in different phases. However, there exist the issues that should be addressed in future works.

- Full dependence on the well written UML diagrams. It is not required by software engineering to use all UML diagrams in every
individual development phase. If using this paper's solution, the cost may go high.

- The granularity of UML activity diagram and activities in process algebra is not easy to be kept exactly same. And applying bi-simulation requires the match of activities and branching structure. It may need certain compromise of precision.

- When applying the algorithm to automate the process of comparison, it is hard to keep the correspondence between the model in the coarser granularity and in the finer granularity due to the complexity of the development phases.

Moreover, the three issues above combined, particularly the third one, could cause the difficulty for the automation of the comparison of the models in different phases due to lack of accuracy. However, the easiness of the application of the algorithm in different software lifecycle may compensate the disadvantages listed above. The further study can be to eliminate the causes of inaccuracy from the algorithm to provide more precise proof and study the applicability of the method being used for software development processes as many as possible.

REFERENCES