ELSEVIER

The Journal of Systems and Software 47 (1999) 105–110

The Journal of Systems and Software

www.elsevier.com/locate/jss

# Consistency management in a process environment

J.-Y.J. Chen [a,*], S.-C. Chou [b,1]

[a] *Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC*

[b] *Department of Information Management, Minghsin Institute of Technology, Hsinfong, Taiwan, ROC*

**Abstract**

Software inconsistency is primarily caused by changes. Changing a software product may cause other products to change. Moreover, changing a part of product (sub-product) usually causes other parts to change too. This paper covers software consistency management supports of advanced process environment research (Aper) by (1) decomposing software products into sub-products and establishing relationships among products and sub-products, and (2) defining trigger processes and consistency conditions in relationships. When a (sub-)product is changed, relationships can be traced to identify the affected ones. Trigger processes then dictate developers to handle the affected ones, which normally need to change accordingly. Meanwhile, consistency conditions should be kept among (sub-)products. Violation of the conditions will result in exceptions, which require handling by developers. © 1999 Elsevier Science Inc. All rights reserved.

*Keywords:* Process-centered software engineering environment (PSEE); Consistency management; Software change

## 1. Introduction

Software processes (software development processes) are becoming more and more complicated. To facilitate their control, *process-centered software engineering environments* (PSEEs) have been developed (Sutton Jr. et al., 1995; Chen, 1997; Jaccheri and Conradi, 1993; Belkhatir and Melo, 1994; Kaiser and Barghouti, 1988; Peuschel and Schafer, 1992; Iida et al., 1993; Bandinelli et al., 1993; Doppke et al., 1998). A PSEE provides a *process language* to represent *process programs*, which can be *enacted* (executed) in the PSEE.

PSEEs manage software products (i.e., documents, including programs) as well as processes. As agreed, software products change frequently. The change may result in ripple effects, because products generally depend on others. Handling *change ripple effects* is essential in software consistency management.

Most PSEEs (Sutton Jr. et al., 1995; Chen, 1997; Jaccheri and Conradi, 1993; Belkhatir and Melo, 1994; Kaiser and Barghouti, 1988; Peuschel and Schafer, 1992; Iida et al., 1993) facilitate handling change ripple effects by using dependency relationships, which can be traced to identify those affected when a product is changed. To further facilitate managing consistency, some PSEEs (Sutton Jr. et al., 1995) define predicates to monitor consistency conditions. Violation of a predicate will result in an exception and cause developers to correct the inconsistency. Nevertheless, consistency management in most current PSEEs suffers from the following shortcomings.

(1) Developers may need to spend too much time in identifying the exact parts to change. When a software product is changed, a PSEE identifies those affected and inform developers to change them. To change a software product, developers should browse the product and identify the exact parts to change. This could take much time, as the size of a software product is normally large.

(2) Handling of intra-product change ripple effects cannot be supported. In most PSEEs, dependency relationships are established between software products. Therefore, handling *inter-product change ripple effects* can be supported. However, changing a part of a product may cause other parts to change, which is called *intra-product change ripple effect*. Handling in this regard does not seem to be supported in most PSEEs.

To overcome the above shortcomings, software products should be decomposed in order to establish both inter- and intra-product relationships. One of the inter-product relationships is product dependency, which can be established among sub-products of different products. When a part of a software product (i.e., sub-product) is changed, a PSEE traces dependency

[*] Corresponding author. E-mail: jychen@csie.nctu.edu.tw
[1] E-mail: lvscchou@ms15.hinet.net

relationships to identify the affected sub-products, and informs developers to change them. Since sub-products are normally much smaller than products in size, developers will not spend too much time in identifying the exact parts to change. Regarding intra-product relationships, including decomposition and invocation relationships and perhaps others, they can be defined among sub-products of the same product. Those relationships facilitate managing intra-product change ripple effects.

We have developed a PSEE called concurrent software process language (CSPL) environment (Chen, 1997; Chen and Tu, 1994a, b; Chen and Chou, 1998). Currently, it is being enhanced to support (1) consistency management, and (2) process evolution (Chou and Chen, 1999). Moreover, the CSPL environment will be ported onto the Internet platform. A new name Aper (advanced process environment research) is thus given to this research. This paper discusses Aper consistency management which, like other techniques (Grundy et al., 1996; Huh, 1998), is facilitated by relationships that can be traced to handle change ripple effects. Aper also monitors *consistency conditions*, which resemble predicates used in other techniques (Sutton Jr. et al., 1995; Decker, 1987; Hsu and Imielinski, 1985). Techniques of Aper consistency management are (1) decomposing software products and managing inter- and intra-product relationships, (2) defining trigger processes in relationships, and (3) keeping consistency conditions. The relationships and trigger processes facilitate managing change ripple effects. And, the consistency conditions monitor consistency among software products.

In the following text, Section 2 gives an overview of the Aper environment. Section 3 describes consistency management in Aper. Section 4 depicts an example. Finally, Section 5 gives the conclusions.

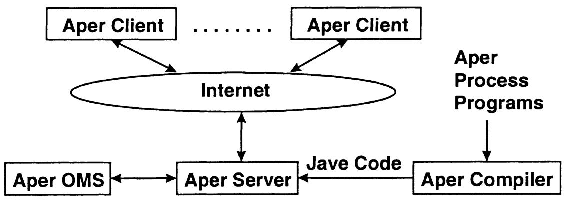

Fig. 1. Apper environment architecture.

## 2. Aper overview

Aper is an Internet-based process environment. It is composed of (1) the Aper language, (2) the Aper compiler, (3) the Aper server, (4) the object management system (Aper OMS), and (5) multiple Aper clients. Aper clients communicate with the Aper server through Internet. See Fig. 1 for the environment architecture.

The Aper compiler compiles process programs to generate Java code which will be enacted (executed) by the Aper server. During process enactment, the Aper server interacts with Aper clients to assign work to developers, and stores and retrieves software products (objects) via the Aper OMS. The Aper language inherits most CSPL syntax (Chen, 1997). However, products are modeled as *Aper classes*. For encapsulation purposes, a class is composed of its specification and body (see Fig. 2). Regarding activities in a process, they are modeled as *Aper activities* (see Fig. 3).

## 3. Aper consistency management

Basic considerations for Aper consistency management are the following.

(1) *Decompose software products and manage inter- and intra-product relationships*: With product decompo-

```
/* class specification */
class DesignDocument is
    /* attributes */
    NonTextType: GraphDocu;
    TextType: TextDocu;
    /* method */
    GenDesign(specification, developer_x,tool_x);
    ....
end DesignDocument;
/* class body */
class body DesignDocument is
    GenDesign(specification, developer_x,tool_x) is
    /* parameter definitions */
    Specification: specification;
    Designer: developer_x;
    Tool: tool_x;
    begin
        perform "Generate the design document." on GraphDocu (W),TextDocu (W) with
                developer: developer_x, reference: specification, tool: tool_x;
    end;
    ....
end DesignDocument;
```

Fig. 2. Aper class.

```
/* activity definition */
activity SystemDesignTask(SystemSpecification, scchou, CASETool) is
/* parameter definiotions */
Specification: SystemSpecification;
Designer: scchou;
Tool: CASETool;
/* data definition */
DesignDocument: design;
begin
     design.GenDesign(SystemSpecification, scchou, CASETool);
     .....
end;
```

Fig. 3. Aper activity.

sition, both inter- and intra-product relationships can be established. The *inter-product relationships*, which are established among sub-products of different products, facilitate managing inter-product change ripple effects. For example, suppose that "sub-product_a1" of "product_a" depends on "sub-product_b1" of "product_b". When "sub-product_b1" is changed, the inter-product dependency relationships can be traced to identify that "sub-product_a1" is affected. Developers will then be informed to change the affected one. Since "sub-product_a1" is expected to be much smaller than "product_a" in size, the time needed to identify the exact parts to change will presumably be reduced.

*Intra-product relationships* facilitate managing intra-product change ripple effects. For example, suppose that "sub-product_1" invokes "sub-product_2". When the latter is changed, the intra-product invocation relationships can be traced to identify that the former is affected. Developers will then be informed to change the affected one. To reduce intra-product change ripple effects, a sub-product can be decomposed into its interface and body. As long as the interface is not changed, changing the body will not affect others.

(2) *Define trigger processes and consistency conditions in relationships*: A *trigger process* defines an event response. Trigger processes defined in relationships are for consistency management. For example, a trigger process in the relationship "DependOn" will inform developers to change those affected when a (sub-)product is changed. Normally, different relationships possess different trigger processes.

*Consistency conditions* define conditions that should be kept among (sub-)products. For example, the specification and design document of a system should possess the same system name. Violation of a consistency condition results in an exception, which will trigger the corresponding exception handler. In Aper, consistency conditions are defined in relationships.

Aper models relationships as relationship types, which are defined in relationship units. Trigger processes and consistency conditions are defined in relationship types. Moreover, various widely-used relationships such as "DependOn" and "PartOf" are built-in relationship types in Aper. In the following subsections, relationship unit, built-in relationship type, and change ripple effect management are respectively described.

### *3.1. Relationship unit*

An Aper relationship unit defines a relationship type. It begins with the keyword "relationship" (see Fig. 4). A relationship type can inherits another by using the keyword "new". For example, in Fig. 4, the relationship type "x_DependOn" inherits the type "DependOn". Each relationship type has two methods, namely "establish" and "dissolve". They are for establishing and dissolving relationships of that type, respectively. For example, the following statement establishes a "x_DependOn" relationship between the products "SystemDesign" and "SystemSpecification".

**x_DependOn.establish(SystemDesign,**
**SystemSpecification);**

One or more trigger processes can be defined in a relationship type. A trigger process begins with the statement "on EventName(parameters) do", where "EventName" is an event and the statements after the "on" statement will be enacted when the event occurs. Aper provides the following events for consistency management: (1) "establish", (2) "dissolve", and (3) "change_n". The first and the second catch events of establishing and dissolving relationships, respectively. For example, the statement "x_DependOn.establish(p1,p2)" will trigger the "establish" event of the relationship type "x_DependOn". Upon that event, the trigger process defined after the statement "on establish(p1,p2) do" will be enacted, where "p1" and "p2" are parameters passed to the trigger process. Thirdly, the "change_n" event catches the event of changing the *n*th (sub-)product in a relationship. For example, the statement "on change_2(p1,p2) do" defines a trigger process that will be enacted when the second (sub-) product in the relationship (i.e., "p2") is changed.

One or more consistency conditions can be defined in a trigger process. A consistency condition begins with the keyword "keep", followed by the condition definition

```
/* Define a relationship type named "x_DependOn", which inherits the built-in type "DependOn" */
relationship x_DependOn is new DependOn with
   establish(product_a, product_b); /* Method to establish a "x_DependOn" relationship between "product_a" and "product b". */
   dissolve(product_a, product_b);  /* Method to dissolve a relationship */

   /* Define the first  trigger process, where the event "change_2" will occur when the second product is changed. */
   on change_2(p1, p2) do
      /* the following statements constitute the trigger process. */
      output "The product",p2,"is changed";
      output "Inform a developer to change the product",p1;

     /* define consistency conditions and exceptions of the trigger process as follows */
     keep
        p1.name=p2.name; /* define the first consistency condition */
     exception
       /* violation of the consistency condition will enact the following statement(s) */
       output "Inform a developer to correct the name inconsistency between ", p1, " and ", p2;
     end; /* End of the first consistency condition */

     keep /* more than one consistency condition can be defined in a trigger process */
        p1.paradigm=p2.paradigm;
     exception
        output "The products ",p1, " and ",p2," are of different paradigms";
        output "Inform a developer to check and correct the condition";
     end;
   end; /* end of the first trigger process */
   on establish(p1,p2) do /* Define the second trigger process */
   . . .
   end;
end; /* end of the relationship type definition */
```

Fig. 4. Aper relationship unit.

(see Fig. 4). If the condition is violated, the exception handler following the keyword "exception" will be enacted to handle the inconsistency. For example, in Fig. 4, when the name consistency (i.e., the condition "p1.name = p2.name") is violated, the trigger process will inform a developer to correct that error.

### *3.2. Built-in relationship type*

Aper provides the following built-in relationship types, in which the first is used for inter-product relationships and the others for intra-product.

(1) The type "DependOn" defines dependency relationships among products or among sub-products of different products. Changing a (sub-)product will enact a trigger process to change those dependent on it. This relationship can be established using the following format, where the former ("product_a") depends on the latter ("product_b").

**DependOn.establish(product_a, product_b);**

(2) The type "PartOf" defines decomposition relationships between a product and its sub-products. If necessary, the relationships can be defined between a sub-product and its sub-products. That means both products and sub-products can be decomposed in Aper. With the relationships, when a (sub-)product is retrieved for manipulation (e.g., browsing, reusing, or updating), its sub-products will also be retrieved. This relationship can be established using the following format, where the former is a sub-product of the latter.

**PartOf.establish(sub-product, product);**

(3) The type "Invoke" defines invocation relationships among (sub-)products. Changing an invoked (sub-)product will enact a trigger process to change the possibly multiple invoking ones. This relationship can be established using the following format, where the former ("product_a") invokes the latter ("product_b").

**Invoke.establish(product_a, product_b);**

(4) The type "InterfaceOf" separates product interface from body. As long as the interface of a (sub-) product remains unchanged, changing its body will not affect others. This idea is directly inherited from information hiding, which dramatically reduces change ripple effects. This relationship can be established using the following format:

**InterfaceOf.establish(product_interface, product_body);**

### *3.3. Change ripple effect management*

Change ripple effects are managed through the cooperation of Aper OMS and Aper server. Aper OMS detects the following events during process enactment: (1) the establishment of a relationship, (2) the dissolution of a relationship, and (3) the change of a (sub-) product. The events detected are then sent to the Aper server, which will enact the trigger processes corresponding to the events. For example, suppose that a "x_DependOn" (see Fig. 4) relationship between "SystemDesign" and "SystemSpecification" has been established by the following statement:

**x_DependOn.establish(SystemDesign,
SystemSpecification);**

When “SystemSpecification” is changed, the Aper OMS will detect the event “change_2” (see Section 3.1 for the definition of the event “change_2”). The event, together with the product names “SystemDesign” and “System-Specification”, will be sent to the Aper server. This will cause the Aper server to enact the trigger process “on change_2”.

The Aper OMS also monitors consistency conditions defined in relationship types. If a condition is violated, the Aper OMS will raise the corresponding exception to inform the Aper server, which will enact the handler of the exception.

## 4. An example

The specification and design document of a simplified supermarket management system is used as an example here. Requirements for the system are described below.

> The supermarket management system manages the stock levels and re-order levels of goods. If the stock level of an item is under its re-order level, the system will order the item from a supplier. The system also manages employee status.

The system’s (sub-)products and relationships are shown in Fig. 5. In Fig. 5, the system specification is decomposed into three sub-specifications, namely “InventorySpec”, “OrderSpec”, and “EmployeeSpec”. Moreover, the design document is decomposed into “InventoryDesign”, “OrderDesign”, and “Employee-Design”, where the former two are further decomposed. The sub-designs of “InventoryDesign” and “OrderDesign” are separated into their interfaces and bodies. For example, “PlaceOrder”, which is a sub-design of “OrderDesign”, is separated into the interface “PlaceOrderInterface” and the body “PlaceOrderBody”. Note that the sub-design “PlaceOrder” is not shown in Fig. 5, because it is composed of its interface and body.

Suppose that the sub-specification “OrderSpec” is changed. Then, the invocation relationship between “OrderSpec” and “InventorySpec” will cause the latter to change accordingly. Note that the change is accomplished by enacting trigger processes defined in the built-in relationship type “Invoke”. Moreover, the dependency relationship (“x_DependOn”) between “OrderSpec” and “OrderDesign” will cause the latter to change by enacting the trigger process “on change_2” defined in the relationship type “x_DependOn” (see Fig. 4). Note that the following two consistency conditions are defined in “x_DependOn”: (1) “p1.name = p2.name” and (2) “p1.paradigm = p2.paradigm”. The former requires that the depending and depended products should have the same name, while the latter requires that those products should be of the same paradigm (e.g., object-oriented and function-oriented).

When changing “OrderDesign”, let us assume that its sub-design “PlaceOrder” is changed. Then, if the interface of “PlaceOrder” (i.e., the document “PlaceOrder-Interface”) is not changed, nothing will be affected. However, if the interface is changed, the document “CheckStockLevelBody” will be affected owing to the invocation relationship. The affected should then be changed by enacting a trigger process defined in the built-in relationship type “InterfaceOf”.

## 5. Conclusions

This paper discusses consistency management in the Aper environment, which is supported by managing change ripple effects using the following techniques: (1) decomposing software products and managing inter- and intra-product relationships, and (2) defining trigger processes and consistency conditions in relationships.

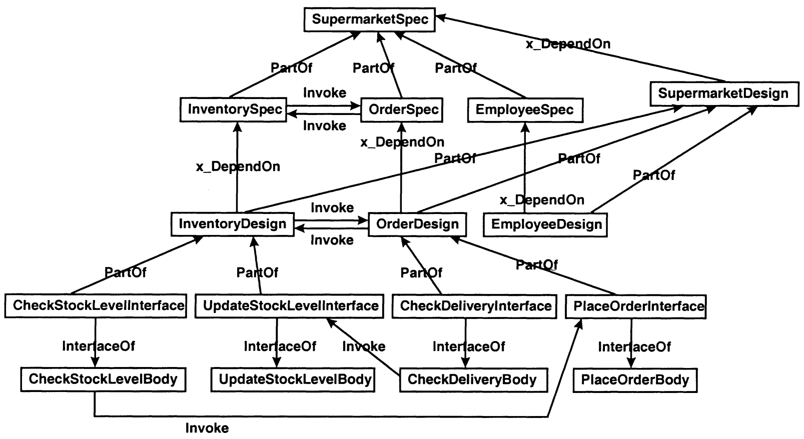

Fig. 5. Relationships between the supermarket specification and design document.

Relationships can be traced to identify those affected when a (sub-)product is changed. Trigger processes can then inform developers to handle the affected ones. And, consistency conditions are monitored to enforce (sub-)product consistency.

In summary, Aper consistency management offers the following features.

(1) Decompose software products to allow establishing inter- and intra-product relationships. With them, handling of both inter- and intra-product change ripple effects can be supported. Moreover, since sub-products can be much smaller than products in size, developers need not spend too much time in identifying the exact parts of a product to change.

(2) Trigger processes can be defined in relationship types to catch and handle events that are relevant to consistency management. In addition, consistency conditions and their exception handlers can be defined to enforce consistency.

(3) Aper relationship unit is capable of defining all kinds of relationships and extending the built-in relationship types. This is expected to increase the flexibility of consistency management.

## Acknowledgements

The authors wish to thank ARES Corporation for their suggestion during our visit there. This research is sponsored by the National Science Council in Taiwan under Grant NSC88-2213-E-009-035.

## References

Belkhatir, N., Melo, W.L., 1994. Supporting software development process in Adele 2. Comput. J. 37 (2), 621–628.

Bandinelli, S.C., Fuggetta, A., Ghezzi, C., 1993. Software process model evolution in the SPADE environment. IEEE Trans. Software Eng. 19 (12), 1128–1144.

Chen, J.-Y., Tu, C.-M., 1994a. An Ada-like software process language. J. System Software 27 (1), 17–25.

Chen, J.-Y., Tu, C.-M., 1994b. CSPL: a process-centered environment. Inform. Software Tech. 36 (1), 3–10.

Chen, J.-Y.J., 1997. CSPL: An Ada95-like, Unix-based process environment. IEEE Trans. Software Eng. 23 (3), 171–184.

Chen, J.-Y.J., Chou, S.-C., 1998. Enacting object-oriented methods by a process environment. Inform. Software Tech. 40(5–6), 311–325.

Chou, S.-C., Chen, J.-Y.J. Process evolution support in concurrent software process language environment. Inform. Software Tech., to appear.

Decker, H., 1987. Integrity enforcement on deductive databases. In: Proceedings of the First International Conference on Expert Database Systems, pp. 381–396.

Doppke, J.C., Heimbigner, D., Wolf, A.L., 1998. Software process modeling and execution within virtual environments. ACM Trans. Software Eng. Methodology 7 (1), 1–40.

Grundy, J.C., Hosking, J.G., Mugridge, W.B., 1996. Supporting flexible consistency management via discrete change description propagation. Software Pract. Experience 26 (9), 1053–1083.

Hsu, A., Imielinski, T., 1985. Integrity checking for multiple updates. In: Proceedings of the ACM-SIGMOD Conference on Management of Data, Austin, pp. 152–168.

Huh, S.-Y., 1998. An object-oriented database model for a change management framework in workgroup computing systems. Inform. Software Tech. 40 (2), 79–92.

Iida, H., Mimura, K.-I., Inoue, K., Torii, K., 1993. Hakoniwa: Monitor and navigation system for cooperative development based on activity sequence model. In: Proceedings of the Second International Conference on the Software Process, IEEE Computer Society, pp. 64–74.

Jaccheri, M.L., Conradi, R., 1993. Techniques for process model evolution in EPOS. IEEE Trans. Software Eng. 19 (12), 1145–1156.

Kaiser, G.E., Barghouti, N.S., 1988. Database support for knowledge-based engineering environments. IEEE Expert 3 (2), 18–32.

Peuschel, B., Schafer, W., 1992. Concepts and implementation of rule-based process engine. In: Proceedings of the 14th International Conference on Software Engineering, pp. 262–279.

Sutton Jr., S.M., Heimbigner, D., Osterweil, L.J., 1995. APPL/A: A language for software process programming. ACM Trans. Software Eng. Methodology 4(3) 221–286.

**Jen-Yen Jason Chen** received a B.S. degree in Industrial Engineering from Tung Hai University, Taiwan; an M.S. degree in Industrial Engineering, an M.S. degree in Computer Science, and a Ph.D. degree (1986) in Computer Science and Engineering from the University of Texas at Arlington. He is now a professor in the Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan. Dr. Chen won Top Scholar in the assessment of system and software engineering scholars in 1995. He specializes in software process programming and environment. And, he is a senior member of IEEE.

**Shih-Chien Chou** received a Ph.D. degree in computer science and information engineering in 1996 from National Chiao Tung University, Taiwan, ROC. Since 1997 he has served as an assistant professor in the Department of Information Management, Minghsin Institute of Technology, Taiwan, ROC. His research interests include software engineering, process-centered software engineering environment, object-oriented analysis and design, and software reuse.