IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4552

Component Context Specification and Representation in System Analysis and Design

Zheying Zhang and Janne Kaipala
Department of Computer Science and Information Systems
University of Jyväskylä, PL 35, FIN-40351 Jyväskylä, Finland
zhezhan@cc.jyu.fi, jka@it.jyu.fi

ABSTRACT

The apparent lack of design information is one of the most significant barriers that system developers face to reuse or change component-based information systems. This paper tackles the problem by presenting and exemplifying the frameworks of component context and its hypertext data model. It addresses the possible linking of contextual knowledge to components, including the conceptual dependencies of component construction, reuse, and implementation, as well as the rationale behind design and reuse processes. Furthermore, it illustrates the hypertext approach to contextual knowledge representation, which provides ways for users to express, explore, recognize, and negotiate their shared context.

1 INTRODUCTION

In order to succeed in collaboration, a component based development (CBD) environment should provide ways for users to express, explore, recognize, and negotiate their shared context. The context is elaborated in terms of the specific application domains and specific intentions, which increase richness of interactions among stakeholders while avoiding repetition, and thereby enhancing reusability.

As Jones [1] indicates only 15% of the requirements for a new system are unique to the system while the remaining 85% comes from requirements of existing systems. Information systems development (ISD) is hereby a process to retrieve and adapt reusable components. The component retrieval process is contextual: a system designer is faced with specific application domains which he looks with some design decisions in mind. The support of the retrieval process requires that knowledge should be provided about contexts in which component can be used. Generally, a component repository does not carry information about the possible use situations, and little semantic support can be provided in the search task. Our view is that knowledge about the use context of components needs to be formalized, stored, and presented with components.

Context exists at various layers of understanding and can be defined and used from many perspectives due to different situations. In a CBD environment, a context forms specific relationship types among components and its development environment [2], such as the conceptual or semantic relationships between components, the domain specific relationships between components and its development environment, and the rationale of component design and reuse [3]. The contextual knowledge will benefit system development activities by increasing available knowledge and richness of interactions between components and users; capitalizing on existing knowledge and previous experience of stakeholders involved in the system development project and decreasing difficulties resulting from individual differences in understanding in a specific application domain.

However, current modeling techniques do not document the reasoning and the rationale behind the suggested solution. The design models are often not communicative, and descriptions cannot be fully understood by users and stakeholders [4]. This makes it difficult to understand models and reuse older ones or parts of them. The purpose of this paper is to present the taxonomy of contextual knowledge for components generated during system analysis and design, and to build a hypertext data model for further implementation. The specification of component context recapitulates our preceding research in component definition [2], at the same time the data model represents solutions to component context definition and utilization.

2 COMPONENT CONTEXT IN SYSTEM ANALYSIS AND DESIGN

Understandability and readability of a component is crucial for its reuse and evolution. While component interface provides essential information, in order to select and reuse a component various forms of contextual knowledge is needed, including the domain description, the conceptual structure and dependency, and the rationale that records predictable and unpredictable ways and environments to facilitate reuse. An extensive context framework has to be developed in order to provide enhanced services in the reuse and the system development processes. The framework must include both the static information derived from component concept and content and the ongoing arguments and decisions which reveal CBD processes. Accordingly, we incorporate the conceptual dependency and the rationale in the component context framework.

2.1 Conceptual Dependency

With the growing number of concepts and the increasing interdependency of ISD methodologies, the components generated from design models become increasingly complex. They are often designed with extremely subtle dependencies on other components that are not explicitly described [5]. However, the construction and maintenance of component-based system analysis and design models require clear understanding of the dependencies between components. Conceptual dependencies between components exist thereafter. The conceptual dependency, as determined by conceptual semantics, forms a way of representing the relationships between components based on a particular meaning of the concepts [6]. It implies the information about how components may and should be used or have been reused in conjunction with other components. There are varied conceptual dependencies with wide variations in the format and content across different contexts, e.g. the definition dependency, the reuse dependency, and the implementation dependency. Clearly, the dependencies between components are necessary and desirable. They need to be clearly expressed by component designers and well understood by users.

Definition Dependency. A definition dependency from a concept CPx to a concept CPy is created if CPx is used in the definition of CPy [7]. In short, it represents part-of relationships among components. The part-of relationships always exist in component-based systems. They are closely interrelated, but difficult to track in a complex system design if the tool does not facilitate the representation of the

definition dependency. For example, a small granularity level component, like the class *PhoneNumber*, is always involved in one (or more) relatively large granularity level component(s), like a class diagram specifying the structure of phone book. Moreover, components can be used as a part of a "larger" component on the same granularity level for a wider or more complete definition, like the relationship between a state transition diagram and the decomposed diagrams therein.

Reuse Dependency. The introduction of component technology into system analysis and design is likely to require a major paradigm shift in design practices in order to better incorporate reuse. Some components are closely related by means of reuse. The reuse dependency identifies the conceptual adaptation between a reusable component and the target ones. In general, it can be divided into "reuse-by-copy" and "reuse-by-reference". Reuse-by-copy occurs when the component user copies (parts of) a component and changes it to meet local needs. Reuse-by-reference on the other hand requires that the same component is used and shared by all users. The reuse dependency indicates the ways to replace or update a component. For example, any modifications on a component will reflect the components having "reuse by reference" dependencies on it, and component users thereby should inspect all reuse related components thoroughly before making any modification.

Implementation Dependency. An implementation dependency describes how a component depends on other components for its implementation [8]. It connects the components that specify the same problem domain or solution but are generated at different development stages. The implementation dependency presents a traceable relationship between components from the high level requirements down to the final implementation, and it thus helps users to trace components' implementation in both forward and backward directions [9, 10]. It enhances reuse by enabling the use of high-level requirement components as the basis to select lower level design or code components [9-11]. Moreover, it enables to analyze the impact of changes in requirements to the rest of the design [10, 11].

The different types of dependencies help designers easily keep track of components at different system development stages, and their interrelationships. Through the definition dependency and reuse dependency component users can see the impacts of possible component update and replacement: "Which other components use this component?" or "Which components are (re)used by this one?". Furthermore, when different types of dependencies have been calculated, it is possible to create a component dependency graph for the design project, by which system designers have the possibility of marking selected components as critical, to indicate that they must not be affected by a component update or replacement, which provides support for component configuration management and change management.

2.2 Rationale

Successful ISD stories address that knowledge from the past and from various stakeholders is used in its processes. Rationale captured in system analysis and design, is one way to keep such knowledge. It includes decisions, alternatives, arguments, and assumptions [11-15]. Generally, the rationale varies widely in quality, formality and in the level of detail across different components and the discussion issues [10]. In a component-based system analysis and design, the rationale records the understanding of why a component has been developed and reused the way it has. More specifically, it records distinct purposes and concerns from component designers and users. On one hand, component design involves decisions and assumptions that drive component reuse, on the other hand, the component reuse process produces feedback that supports component maintenance and system evolution [16, 17]. While considering the different purposes of component context, we distinguish between the design rationale and the use rationale.

Design Rationale. In practice, the design rationale recorded by component designers serves as an externalization of a design for both component users and designers, while it communicates design information to them. Consequently, design rationales provide domain information and answer specific questions of component definition and maintenance. A well-designed component can be easily reused in different design scenarios within the same domain. Therein, design rationale is an ideal way for component designers to record the assumptions of diverse design scenarios and to supervise the component reuse process by suggesting component users that where and how the component should be modified to meet a new set of requirements.

Use Rationale. In general, component users capture use rationale in the sequential stages of a component reuse process: search, selection, adaptation and integration [3]. Accordingly, use rationales can be categorized according to these stages. The rationale under this context has two major functions. On one hand, it reflects the scenarios of using a component, on the other hand, it evaluates the component complexity and reusability and provides feedback for component maintenance.

The loss of design information becomes a problem when systems have to change to meet unanticipated requirements. The engineers responsible for system evolution must analyze the design and infer the reasons for particular design choices. As a reference to the system engineers, the rationale identifies the justification for the evolution of systems by recording component design and reuse activities as the arguments for and against the alternatives and the final decisions, which will significantly decrease the cost of system evolution [16, 17]. Moreover, a chronological display of the rationales provides clear understanding of the component evolution process, facilitating maintenance and reuse.

3 LINK TYPES FOR CONTEXTUAL INFORMATION REPRESENTATION

Hypertext is a feasible approach to model integration and context representation [11, 18-24] especially for the rationale that can be captured any time at the analysis and design stage. It represents and integrates contextual information by means of nodes and links. Nodes represent components and their contextual information while links represent their relationships. These links can carry different types of information, such as the conceptual dependencies and the rationale. Accordingly, we distinguish between three types of traceability links: association, annotation and debate links, as shown in Table 1. These links are applied according to the link types presented by Oinas-Kukkonen [23], and we further consider the reuse subtype links.

Association Links are defined mainly to represent the conceptual dependencies between components. In practice, they represent relationships between related artifacts such as components, documents, a part of a component, and a part of a document. They are generated to let users perform several tasks during design: track the composition of components, track the modification and refinement history of a component, manage the repercussions of changes in one on other components that reuse and depend on it [10], identify the requirements related to components, and ensure consistency between the components in the successive stages of the life cycle. Due to the diverse roles which association links take in different contexts, further link semantics can be expressed by the subtypes, like the definition dependency (e.g. is-part-

***Table 1**. Types of links for component context representation*

Traceability Link Type	Subtypes	Contextual knowledge
Association	Definition dependency: is-part-of Reuse dependency: is-reused-by-{copy, reference} Implementation dependency: is-implemented-by	Conceptual dependency
Annotation	Design rationale: design rationale Use rationale: rationale-for-component-{search, selection, adaptation, integration}	Rationale
Debate		

of), the reuse dependency (e.g. reuse-by-copy) and implementation dependency (e.g. implements).

Annotation Links provide a way to connect information, namely an annotation node, to a (part of the) component or a (part of the) document. In essence, they support reuse processes by capturing information related to a particular design situation, and component reuse and maintenance process. Annotation links lead to annotation nodes that allow free text representation, thereby enabling the recording of any information that demands text representation, including textual design rationale. The more structured approach to capturing arguments over design decisions is provided by debate links.

Debate Links represent the argument-based rationale behind components including reasons for evolutionary steps and contextual information in the design and reuse phases. They integrate the stakeholders' arguments to the design components. In order to provide a clear overview of the captured rationale, we can further categorize rationale in line with phases of the life cycle and the aspects of the issues to be argued.

4 HYPERTEXT DATA MODEL FOR CONTEXTUAL INFORMATION REPRESENTATION

In terms of component-based reuse, the hypertext data model has two kinds of nodes: component nodes (or design elements inside a component) and contextual information nodes (e.g. Questions, Answers, Arguments and Annotations). As shown in Fig. 1, both types of nodes can be sources or targets of links. Moreover, the link source can be specified as a piece of text inside a node. As stated in section 3 we propose traceability links to connect the component nodes and contextual information nodes. Different types of traceability links are used under different contexts.

Arrows in Fig. 1 demonstrate the information flows that are traceable between two types of nodes. In detail, the source of an annotation link can be any node and the target is an annotation node. Similarly, a debate link can start anywhere and the link target is a debate node. The source and target of an association link can be any node.

Links have attributes such as subtype, keywords, creator, creation time, which can be used to define further link semantics for reuse purposes. The contextual information nodes can be organized into named collections: Questions belong to "debate spaces" and Annotations belong to "annotation spaces". Furthermore, debate nodes are linked by using specific link types: Questions are linked to Answers by "answers-to-question", and Arguments are linked to Answers by "supports" or "objects-to" links.

In order to demonstrate the representation of component context, we discuss a mobile phone user interface design scenario within the phone product family. Suppose that the company has created a mobile phone user interface design architecture, domain models, and a set of reusable components to quickly deliver mobile phones in various versions. In this scenario, the company plans to deliver a user interface design of mobile phone version 4.4 by reusing components from the prior 4.2 version. Compared with version 4.2, a new feature in version 4.4 is an improved calendar that provides a "structured date editor". Accordingly, the requirements list of version 4.4 is created by reusing the requirements list version 4.2. Meanwhile, new requirements are added, including the requirement "The Calendar shall support structured date input". As shown in Fig. 2, in the process of creating the new requirements list, traceability links such as reuse dependency (arrow 1) have been created, and the implementation dependency (arrow 2) was traced to get the corresponding design model components of version 4.2. Hereby, the design models are reused by copy (arrow 3), and a new implementation dependency is created between the requirements list and the design model components (arrow 4). At the same time, structured date input function is added into the *Calendar* component, and the rationale about the different implementation alternatives and decision related to the structured date input function are recorded by the annotation link (arrow 6) and the debate link (arrow 7).

In this brief scenario, information is captured by creating different types of contextual links during the analysis and design processes. The links represent various conceptual dependencies and rationales. Although the presented scenario is simple and uncompleted, as compared with the phone development life cycle, masses of contextual information and 14 contextual links have been created. Without the context specification and its hypertext representation, the substantial knowledge about the component logic and semantic structure and about the design rationale is buried in the development process and is difficult to remember, retrieve, and reuse after the project.

Fig. 1. *The hypertext data model*

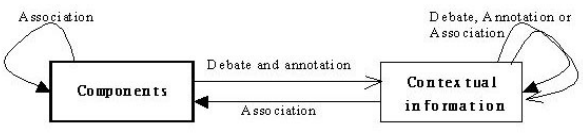
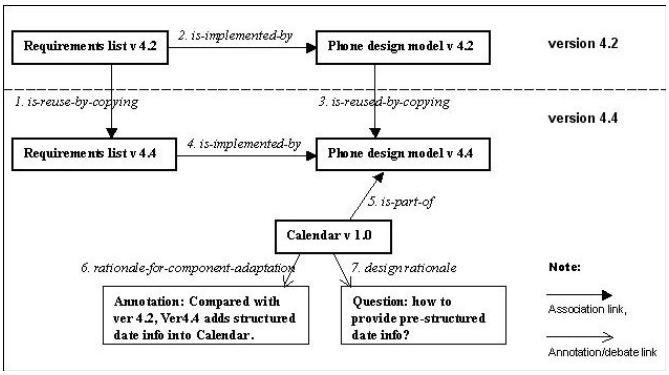


Fig. 2. *Examples of different traceability links in a mobile phone user interface design scenario*



5 CONCLUSIONS

Component context drives reuse activities from the requirements analysis towards the final implementation in a CBD environment. However, the benefits of component context will not come to full fruition unless they are elaborately defined and directly integrated into the basic development activities of ISD [25]. In this paper we have tried to increase the understanding of component context in perspectives of conceptual dependencies and rationales that supports CBD. The different types of dependencies and rationales assemble the different aspects of component context, which constitutes the conceptual foundation of component reuse. They are provided as an adjunct part of a component and embedded in the CBD processes. Furthermore, the mechanisms of component context representation and the approach to facilitating the contextual knowledge are proposed and demonstrated by showing the hypertext data model which consists of two types of nodes (component and contextual information) and three types of traceability links (the association link, the annotation link, and the debate link). The hypertext data model represents concepts and mechanisms to facilitate the representation of syntax and semantics of components and the contextual information between components and its design environment. We believe that once it is integrated to a CASE tool, the hypertext supported CASE tool can better support information tracing and the interaction between components and stakeholders. We expect this to alleviate the difficulties resulting from individual differences in understanding the system architecture and its components in a specific application domain and to make CBD more practical and effective.

REFERENCES

1. Jones, T.C., Reusability in Programming: A Survey of the State of the Art. IEEE Transactions on Software Engineering, 1984. 10(1).
2. Zhang, Z. Defining Components in a MetaCASE Environment. Proceedings of the 12th Conference on Advanced Information Systems Engineering (CAiSE*00). 2000. Stockholm, Sweden: Springer.
3. Zhang, Z. and K. Lyytinen, A Framework for Component Reuse in a Metamodelling based Software Development. Requirements Engineering Journal, 2001. 6(2): p. 116 - 131.
4. Bubenko, J.A. Challenges in Requirements Engineering. Invited talk at the Second IEEE International Symposium on Requirements Engineering. 1995.
5. Edwards, A., et al. Software Component Relationships. Proceedings of the eighth Annual Workshops on Institutionalizing Software Reuse (WISR8). 1997. Columbus, OH.
6. Schank, R.C., Conceptual Dependency: A Theory of Natural Language Understanding. Cognitive Psychology, 1972. 3(4): p. 532 - 631.
7. Castellani, X. Overviews of Models Defined with Charts of Concepts. IFIP WG8.1 International Conference on Information System Concepts: An Integrated Discipline Emerging. 1999. Leiden, the Netherlands.
8. Whittle, B., Models and Languages for Component Description and Reuse. ACM SIGSOFT, 1995. 20(2): p. 76 - 87.
9. Jarke, M., Requirements Tracing. Communications of the ACM, 1998. 41(12): p. 32 - 36.
10. Ramesh, B. and M. Jarke, Towards Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, 2001. 27(1): p. 58 - 93.
11. Barber, K.S., et al., Requirements Evolution and Reuse Using the Systems Engineering Process Activities (SEPA). Australian Journal of Information Systems, 2000. 7(1): p. 75 - 97.
12. Lee, J. and K. Lai, What's in Design Rationale. Human Computer Interaction, 1991. 6(3-4): p. 251 - 280.
13. Perry, D.E. and A.L. Wolf., Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 1992. 17(4): p. 40 - 52.
14. Tracz, W., L. Coglianese, and P. Young, A Domain-Specific Software Architecture Engineering Process Outline. ACM SIGSOFT Software Engineering Notes, 1993. 18(2): p. 40 - 49.
15. Sommerville, I. and P. Sawyer, Requirements Engineering: A Good Practice Guide. 1997: John Wiley & Sons. 391.
16. Monk, S., et al. Supporting Design Rationale for System Evolution. Proceedings of the Fifth European Software Engineering Conference. 1995.
17. Bratthall, L., E. Johansson, and B. Regnell. Is a Design Rationale Vital When Predicting Change Impact? - A Controlled Experiment on Software Architecture Evolution. Proc. Conference on Product Focused Software Process Improvement (PROFES'2000). 2000. Berlin: Springer-Verlag.
18. Bailin, S.C., et al. KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale. Proceedings of the 5th Annual Knowledge-Based Software Assistant Conference. 1990.
19. Creech, M.L., D.F. Freeze, and M.L. Griss. Using Hypertext in Selecting Reusable Software Components. Hypertext 1991. 1991.
20. Sutcliffe, A. Requirements Rationales: Integrating Approaches to Requirements Analysis, Designing Interactive Systems: Processes, Practices, Methods, & Techniques. Proceedings of DIS'95. 1995: ACM Press.
21. Robbins, J.E., D.M. Hilbert, and D.F. Redmiles. Software Architecture Critics in Argo. 1998 International Conference on Intelligent User Interfaces. 1998. San Francisco, CA, USA.
22. Mannion, M., et al. Reusing Single Requirements From Application Family Requirements. 21st IEEE International Conference on Software Engineering (ICSE'99). 1999.
23. Oinas-Kukkonen, H., Improving the Functionality of Software Design Environments by Using Hypertext, Department of Information Processing Science. 1997, University of Oulu: Finland. p. 130.
24. Kaipala, J., Integrating MetaCASE Environments by Using Hypertext - Conceptual, Functional and User Interface Considerations in MetaEdit+, Department of Computer Science and Information Systems. 1999, University of Jyväskylä: Finland. p. 114.
25. Keller, R.K. and R. Schauer. Design Components: Towards Software Composition at the Design Level. International Conference on Software Engineering. 1998: IEEE Computer Society.

An Optimal Policy with Three-Parameter Weibull Distribution Deterioration, Quadratic Demand, and Salvage Value Under Partial Backlogging
Trailokyanath Singh, Hadibandhu Pattanayak, Ameeya Kumar Nayakand Nirakar Niranjan Sethy (2018). *International Journal of Rough Sets and Data Analysis (pp. 79-98).*
www.irma-international.org/article/an-optimal-policy-with-three-parameter-weibull-distribution-deterioration-quadratic-demand-and-salvage-value-under-partial-backlogging/190892