

Modeling Use Cases with the Borland Suite of Tools

By Serge Charbonneau, Principal Consultant for Xelation Software Corporation
(www.xelation.com)

Introduction

Use case modeling is a widely used technique to describe the functional requirements of a system in terms of its behavior. This technique leverages both graphical and textual representations for describing the interaction between users of a system and the system itself.

The goal of this white paper is to demonstrate that the Borland® Suite of Tools can automate use case modeling activities. This paper reviews the techniques of use-case modeling, presents an example based on an Automated Teller Machine (ATM) system and presents how use case modeling techniques can be implemented using the Borland Suite of Tools.

Review of Use Case Modeling Techniques

A use case model is a visual representation of a system's behavior. It is based on two main Unified Modeling Language (UML) visual modeling constructs: actors and use cases.

An actor is someone or something that is outside the system and that interacts with the system. According to this definition, an actor can represent a human being or a machine (device, computer or other). Figure 1 shows the UML representation of an actor.



Figure 1 A UML Actor

A use case is a sequence of actions that yields a result of observable value to an actor. The sequence of actions can be described using a series of textual flows and/or scenarios, using UML activity diagrams or using a combination of both. Figure 2 shows the UML representation of a use case.

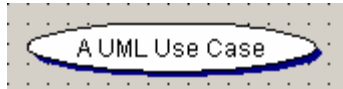


Figure 2 A UML Use Case

Actors and use cases are link together using associations. A unidirectional association from an actor to a use case identifies that the actor can initiate the behavior described in the use case when interacting with the system. In this type of association, the actor is said to be an active actor. A bidirectional association between an actor and a use case identifies that the actor can participate in the behavior described in the use case without initiating it. In this type of association, the actor is said to be a passive actor. A common misconception about bidirectional and unidirectional associations is that they represent the direction the data flows between actors and the system. They don't represent the directionality of data flow. Data can flow both ways. Figure 3 shows an active actor and a passive actor associated with a use case.

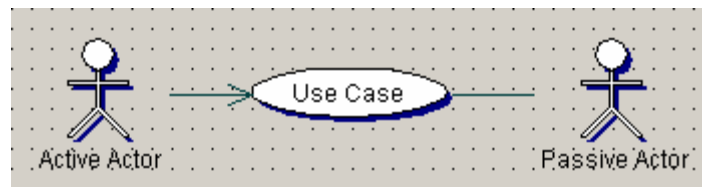


Figure 3 A Passive Actor and an Active Actor Associated with a Use Case

Three types of relationships exist between use cases:

1. Include;
2. Extend; and
3. Generalization.

Figure 4 shows examples of use case relationships.

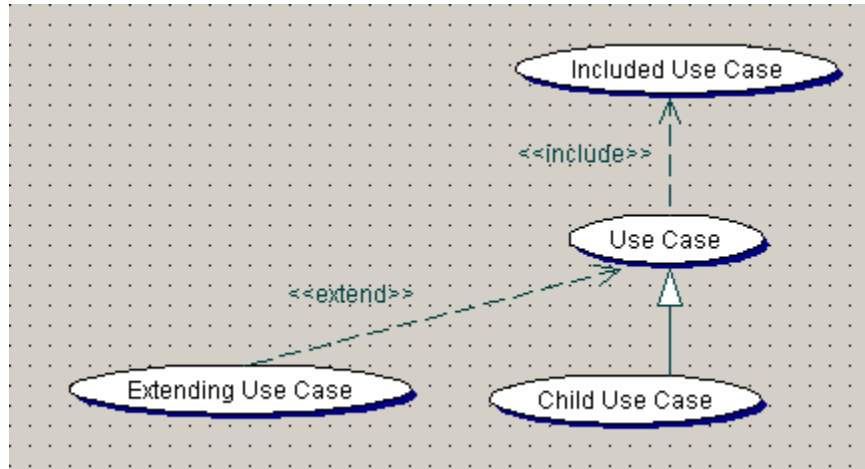


Figure 4 Use Case Relationships

The include relationship shows that the Included Use Case behavior is being invoked from within the Use Case. The extend relationship shows that the Extending Use Case extends the Use Case behavior at a specific extension point if the extension point condition is met. The generalization relationship shows that the Child Use Case inherits all the behavior of the Use Case and extends it with extra behavior.

The only type of relationship that exists between actors is the generalization. Figure 5 shows an example of an actor generalization relationship.

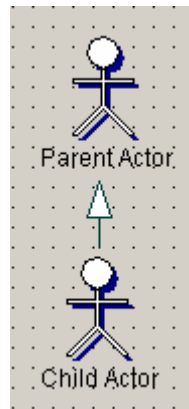


Figure 5 Actor Relationships

The following activities are performed when applying use case modeling techniques to describe the behavior of a system:

1. Draw a system boundary;
2. Create an actor outside the system boundary to represent every role interacting with the system being modeled and every system integrating with the system being modeled;
3. Using the actors previously identified, create a use case for each portion of behavior that will provide value to an actor;
4. Associate actors with use cases using unidirectional and/or bidirectional associations;
5. Refactor use case diagram by leveraging include, extend and generalization relationships as needed to abstract common behavior, optional behavior, exceptional behavior, and behavior that is to be developed in later iterations; and
6. Describe each use case in details.

A use case is described in terms of the following elements:

1. Name: Name of the use case;
 2. Brief Description: Summary description of the use case;
 3. Main/Basic Flow or Primary Scenario: Most common path through the use case;
 4. Alternative/Exceptional Flows or Secondary Scenarios: Less common paths through the use case;
 5. Special Requirements: Non-functional requirements specific to the use case;
 6. Pre-Conditions: State of the system before the execution of the use case;
 7. Post-Conditions: State of the system after the execution of the use case;
- and

8. Extension Points: Point where a use case flow or scenario can be extended.

A flow is a portion of a use case instance while a scenario is a use case instance from start to finish. Flows and scenarios can be described textually or visually using activity diagrams. Figure 6 shows the differences between flows and scenarios for the same use case.

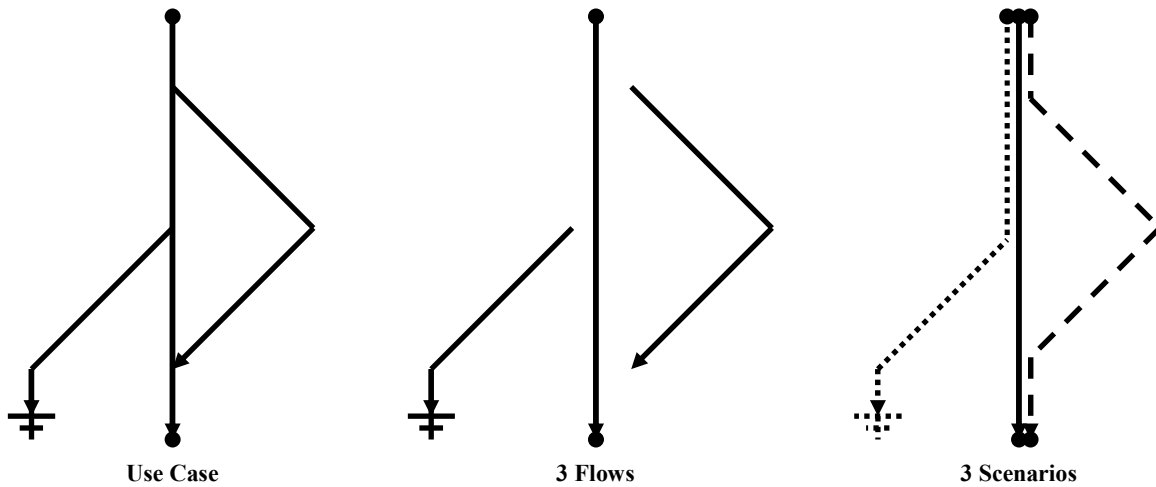


Figure 6 Differences between Flows and Scenarios for the Same Use Case

Use Case Model Example

The example presented in this section applies use case modeling techniques to describe the behavior of an ATM system. This classic example is often used in the literature since it is based on a technology that all readers are familiar with. It has the advantage of making the readers focus on the use case modeling techniques instead of on the ATM system technology.

The main actors interacting with the ATM system are the following:

1. **Customer:** This actor owns at least one bank account and has access to the ATM system using a debit card;
2. **Bank System:** This actor maintains all information about its customers, their accounts, and the transactions performed on their accounts using ATM systems or other means;
3. **Maintenance Representative:** This actor is responsible for general maintenance of the ATM system including repairs; and
4. **Bank Representative:** This actor is responsible for removing envelopes from the ATM system and refurbishing the ATM system with cash.

Other actors could be identified but we will limit the list to the ones identified above for the purpose of this example.

The main use cases describing the behavior of these 4 actors interacting with the ATM system are the following:

1. **Withdraw Cash:** This use case describes how the customer withdraws cash from the ATM system;
2. **Deposit Funds:** This use case describes how the customer deposits funds into the ATM system;
3. **Transfer Funds:** This use case describes how the customer transfers funds using the ATM system;
4. **Pay Bill:** This use case describes how the customer pays bills using the ATM system;
5. **Print Statement:** This use case describes how the customer prints a statement of his or her account using the ATM system;
6. **Remove Envelopes:** This use case describes how the bank representative removes envelopes from the ATM system;
7. **Refurbish Cash:** This use case describes how the bank representative refurbish cash into the ATM system; and
8. **Perform Repairs:** This use case describes how the maintenance representative performs repairs on the ATM system.

Other use case could be identified but we will limit the list to the ones identified above for the purpose of this example.

Figure 7 shows the use case diagram with the system boundary, actors, use cases and their relationships for the ATM system example.

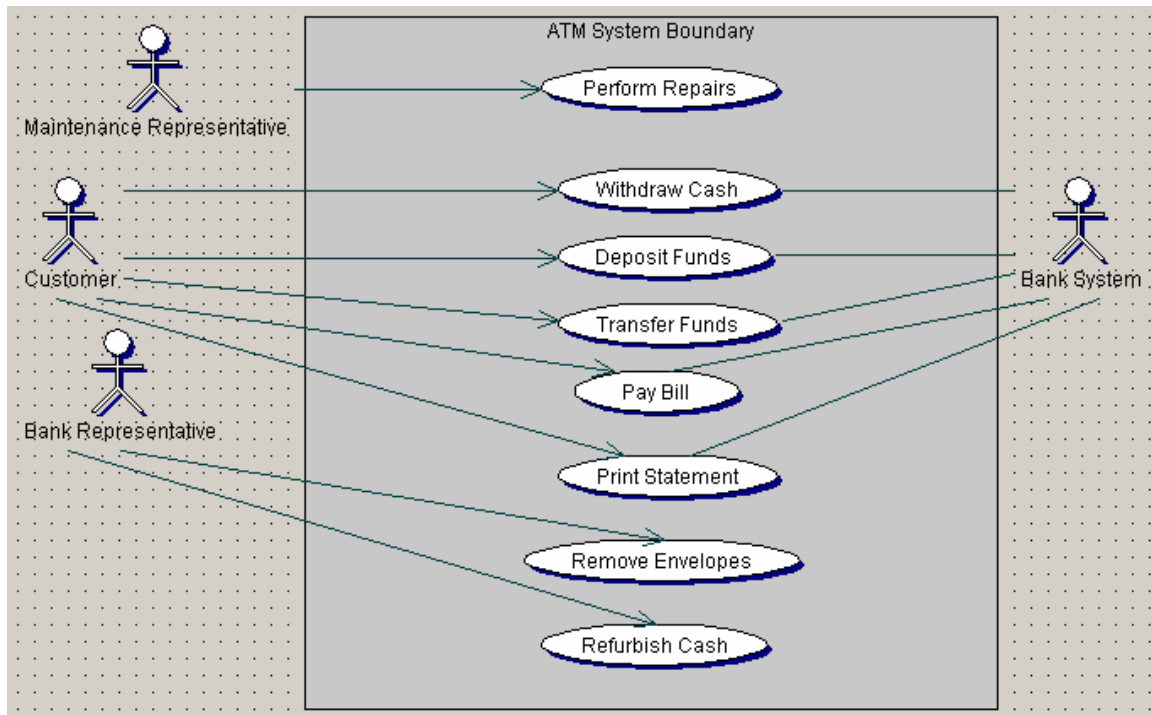


Figure 7 ATM System Use Case Diagram with System Boundary, Actors, Use Cases and their Relationships.

After going through a refactoring exercise, the following use cases are uncovered:

1. Authenticate User: This use case is included by all other use cases where authentication is required;
2. Withdraw Cash from Checking Account: This use case inherits (generalization) all the behavior of the Withdraw Cash use case and specializes it for a checking account;
3. Withdraw Cash from Savings Account: This use case inherits (generalization) all the behavior of the Withdraw Cash use case and specializes it for a savings account;
4. Perform Card Reader Repairs: This use case extends the behavior of the Perform Repairs use case for the ATM system card reader;
5. Perform Cash Dispenser Repairs: This use case extends the behavior of the Perform Repairs use case for the ATM system cash dispenser;
6. Perform Envelope Collector Repairs: This use case extends the behavior of the Perform Repairs use case for the ATM system envelope collector; and
7. Perform Printer Repairs: This use case extends the behavior of the Perform Repairs use case for the ATM system printer.

The following actors are also uncovered during the refactoring exercise:

1. Teller: This actor inherits (generalization) all the properties of the Bank Representative actor and specializes it.
2. Manager: This actor inherits (generalization) all the properties of the Bank Representative actor and specializes it.

Figure 8 shows portions of the use case diagram affected by the refactoring. In order not to clutter the original use case diagram, it is often a good practice to describe the entire use case model in more than one use case diagram. These diagrams should be linked or nested when needed.

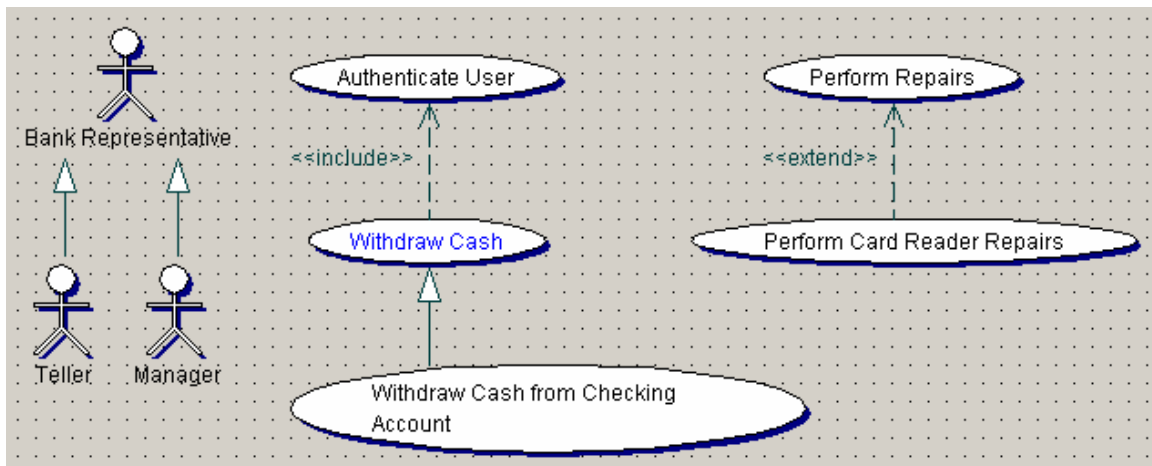


Figure 8 Portions of the ATM System Use Case Diagram Affected by the Refactoring.

The Withdraw Cash use case description with all its detailed elements is the following:

1. Name: Withdraw Cash.
2. Brief Description: This use case describes how the customer withdraws cash from the ATM system.
3. Main Flow
 - a. This use case starts when the customer is authenticated as described in use case “Authenticate User”.
 - b. The system prompts the customer to select one of the following operations:
 - i. Withdraw Cash;
 - ii. Deposit Funds;
 - iii. Transfer Funds;
 - iv. Pay Bills; and
 - v. Print Statement.
 - c. The customer selects the withdraw cash option.

- d. The system prompts the customer to select one of the following account:
 - i. Checking Account;
 - ii. Savings Account; and
 - iii. Credit Margin Account.
 - e. The customer selects an account.
 - f. The system prompts the customer to enter an amount.
 - g. The customer enters an amount.
 - h. The system prompts the customer if he or she wants to perform another operation.
 - i. The customer selects not to perform another operation.
 - j. The system returns the card to the customer.
 - k. The customer takes the card.
 - l. The system dispenses cash to the customer.
 - m. The customer takes cash.
 - n. The system prints a receipt.
 - o. The customer takes the receipt.
 - p. This use case ends.
4. Alternative Flow 1: Account Balance Too Low - There is not enough money in the customer account to provide the customer with the requested amount.
 5. Alternative Flow 2: ATM System Balance Too Low - There is not enough money in the ATM system to provide the customer with the requested amount.
 6. Special Requirement 1: Currency - The system shall provide cash only in US Currency.
 7. Special Requirement 2: Currency Unit – The system shall provide cash amount in multiples of 20 dollar bills.
 8. Pre-Condition: The customer has at least one active account with the bank.
 9. Post-Condition: The amount withdrawn by the customer is subtracted from the customer account balance.
 10. Extension Point: None.

Figure 9 shows an activity diagram describing visually the sum of all the flows and scenarios of the Withdraw Cash use case.

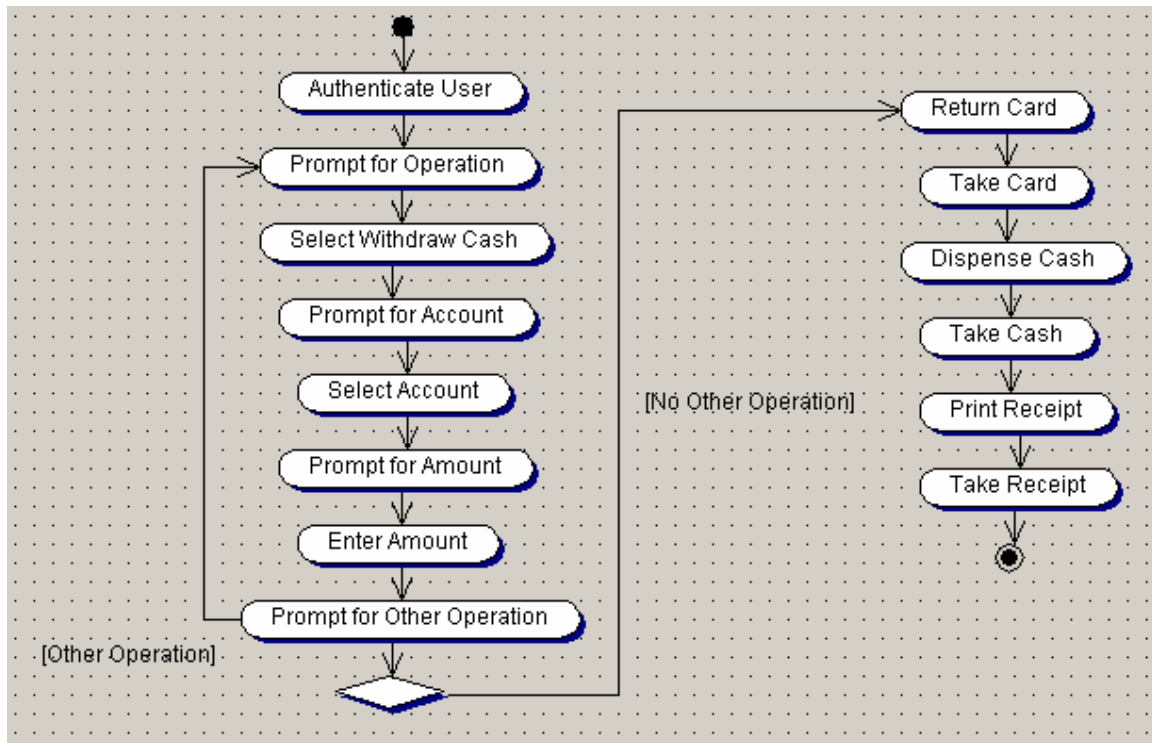


Figure 9 Activity Diagram of the Withdraw Cash Use Case

Use Case Model Example Built with the Borland Suite of Tools

The Borland Suite of Tools includes two tools that provide capabilities for automating use case modeling techniques: Borland CaliberRM™ and Borland Together®.

Borland CaliberRM

Borland CaliberRM is a requirements management tool that allows analysts to capture and manage requirements. It allows the association of attributes with requirements which facilitates the objective management of these requirements. It also allows the creation of traceability links between requirements that are related which helps insure completion of artifacts (lower-level requirements, design elements, test cases, etc.) derived from these requirements. It also enables objective impact analysis when changes are requested on baselined requirements. Finally, it provides baselining capabilities with complete audit trails to make parallel development possible.

The details of a use case can be captured in a document where each section describes a detailed element of the use case: name, brief description, main flow, etc. Or these details can be captured as a hierarchy of requirements. Borland CaliberRM provides capabilities to use one of these two approaches or a combination of both. Using the Borland Document Factory application available with Borland CaliberRM, it is possible to extract the use case requirements from the Borland CaliberRM repository to populate a use case specification template.

One of the main benefits of having use case details as a hierarchy of requirements in Borland CaliberRM is that it allows traceability at any level of granularity. For instance, traceability links can be created between each flow or scenario and test cases hence insuring complete test coverage of a use case.

Another benefit is that it facilitates the allocation of use case detail realizations to different project iterations. For instance, the main flow and two alternative flows of a use case can be realized in iteration “x” while the remaining alternative flows can be realized in iteration “x+1”.

Figure 10 shows the Withdraw Cash use case details organized in a hierarchical fashion described using Borland CaliberRM. It also shows other types of requirements including features and supplementary specification.

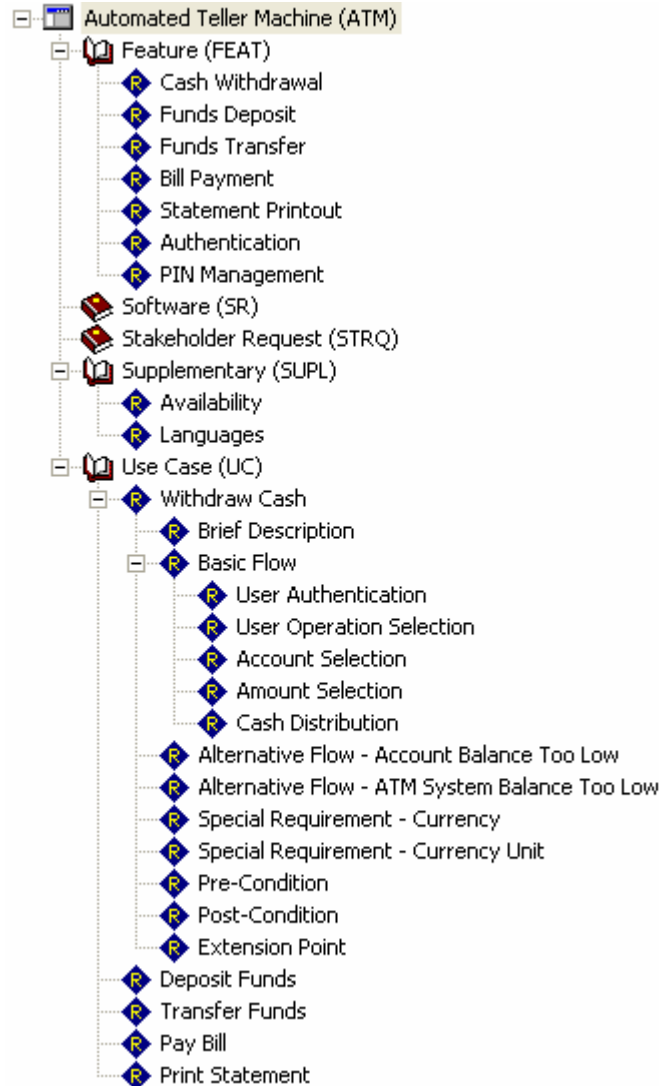


Figure 10 Withdraw Cash Use Case details in Borland CaliberRM

As previously mentioned, Borland CaliberRM allows the association of attributes with requirements to assist project managers in managing the scope of a project. The default attributes included with all requirement types are:

1. Priority;
2. Status;
3. Owner; and
4. Others.

Extra attributes can be created in Borland CaliberRM. For the Use Case requirement type, an extra attribute should be created to identify the type of detail the use case requirement represents. This attribute should be a list of values called “Property” with the following values:

1. Name;
2. Brief Description;
3. Main Flow;
4. Alternative Flow;
5. Flow Step;
6. Special Requirement;
7. Pre-Condition;
8. Post-Condition; and
9. Extension Point.

Figure 11 shows the “Property” attribute with its list of values for the Use Case requirement type.

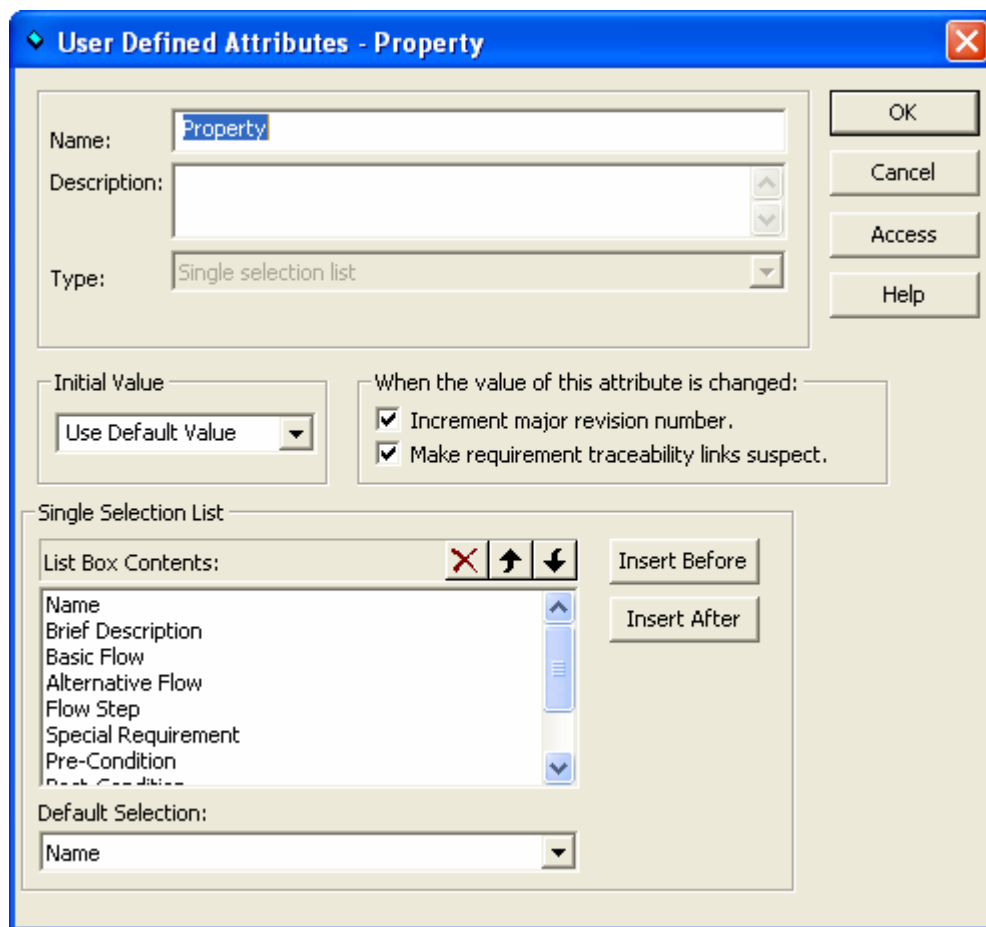


Figure 11 “Property” Use Case Requirement Attribute List of Values

Figure 12 shows the traceability between the Cash Withdrawal feature and the Withdraw Cash use case details.

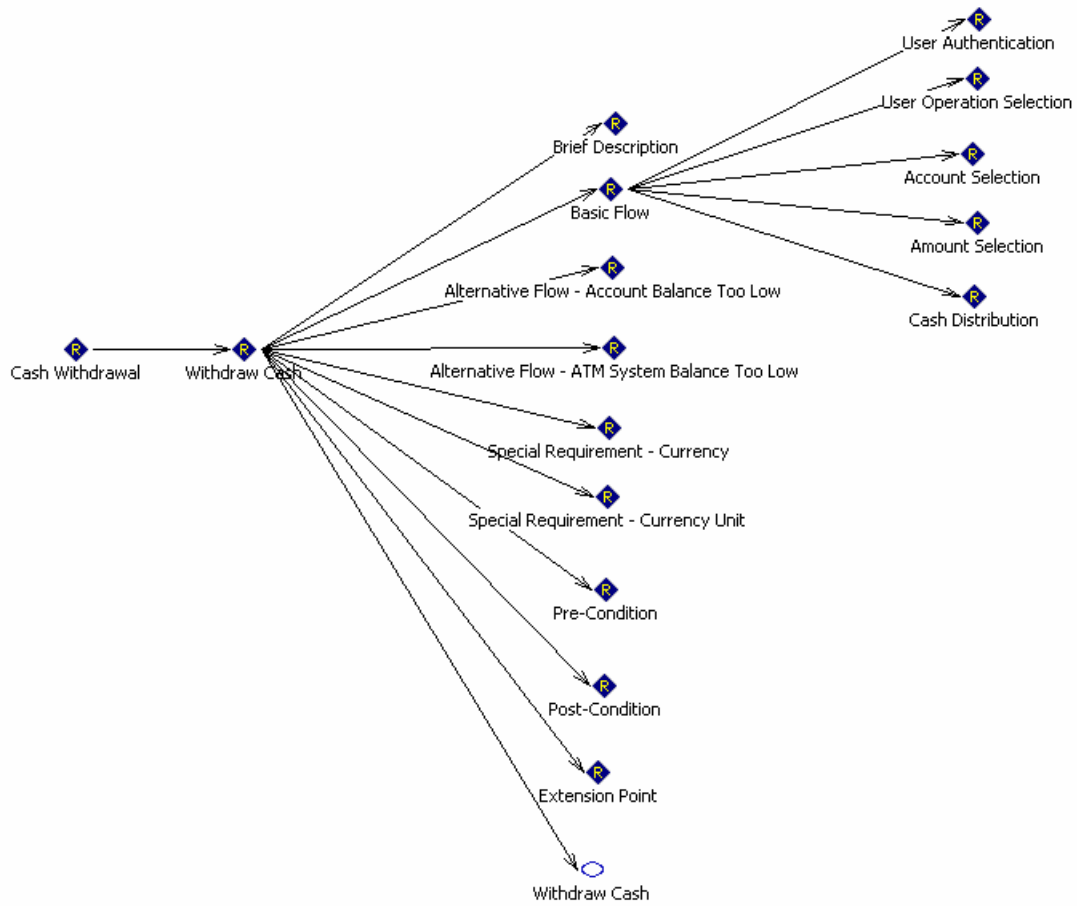


Figure 12 Traceability between the Cash Withdrawal Feature and the Withdraw Cash Use Case Details

Borland Together

Borland Together is a visual modeling tool that allows analysts, architects, designers and developers to visually describe the behavior and structure of a system using the UML. It also allows round-trip engineering between visual elements and source code.

Borland Together allows the creation of use case diagram with actors and use cases, activity diagrams to describe the flows and scenarios of use cases, and traceability links between visual use cases in Borland Together and use case requirements in Borland CaliberRM. All UML diagrams presented in this article were built using Borland Together.

Figure 13 shows traceability between use cases in Borland Together and use case requirements in Borland CaliberRM.

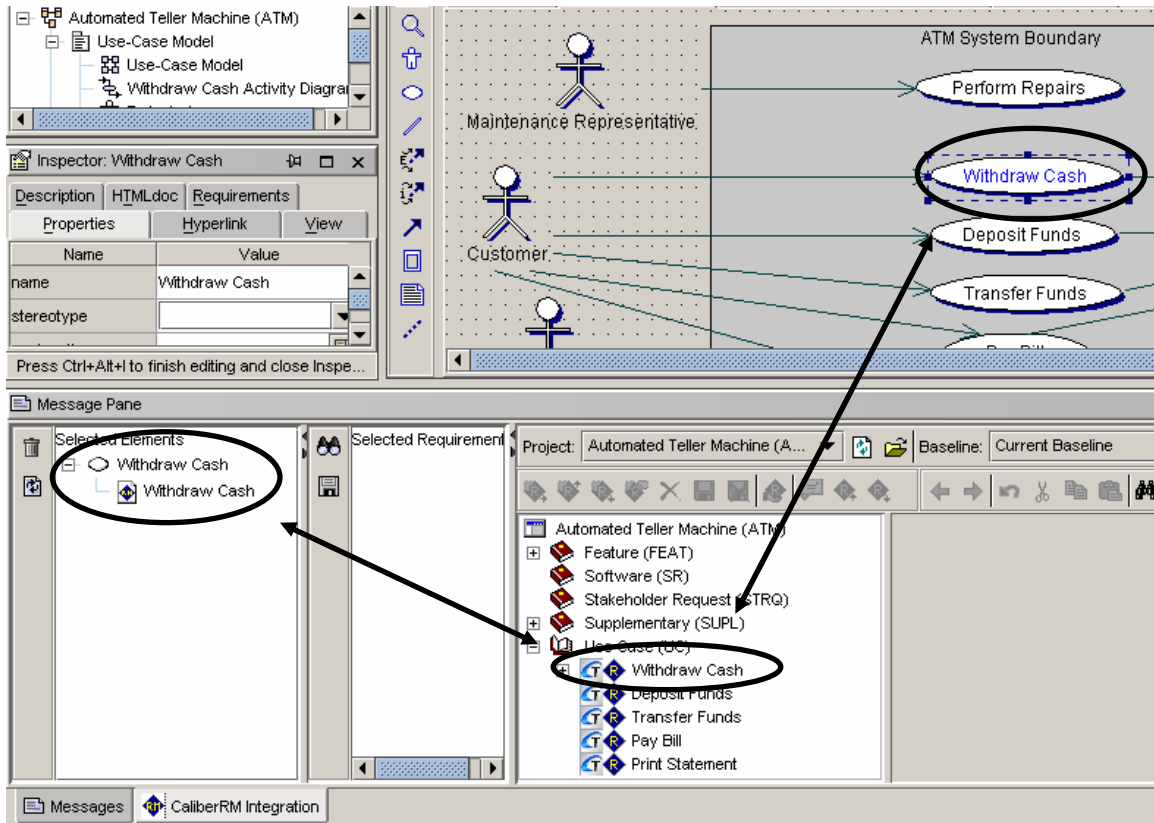


Figure 13 Traceability between Borland Together Use Cases and Borland CaliberRM Use Case Requirements

Conclusion

This paper demonstrated that the Borland Suite of Tools can indeed be used to automate use case modeling activities. Borland CaliberRM supports the creation of use case requirements to describe the details of use cases. Borland Together supports the visual modeling of use case diagrams, and the creation of traceability links between visual use cases and their requirements counterparts.

About Xelation Software Corporation

Xelation Software Corporation is a company founded in 2003 that provides its customers with products and services in the following disciplines of software engineering:

- Requirements Management;
- Architecture;
- Object-Oriented Analysis and Design;
- Development;
- Quality Control;
- Project Management;
- Configuration and Change Management; and
- Process Engineering.

Xelation Software Corporation's mission is to accelerate the success of its customers in reaching their software engineering capability improvement goals.

Xelation Software Corporation has developed a Borland Tool Mentors plug-in for the IBM[®] Rational Unified Process[®] (RUP[®]). To learn more about this product, visit the Xelation Software Corporation web site at: www.xelation.com.

About the Author

Serge Charbonneau is a principal consultant for Xelation Software Corporation. Mr. Charbonneau has over 12 years of experience in the field of software engineering. He has played many roles in software projects and organizations including developer, designer, analyst, team lead and technical representative. He has a solid expertise in software project management, requirements management, architecture, analysis and design, and development processes.