Pre-publication version. To appear in *International Journal of E-Business Research*, Vol. 1, No. 3

Business Process Modeling with URN

Michael Weiss

School of Computer Science
Carleton University
Ottawa, ON, K1S 5B6 (Canada)

phone: +1 (613) 520-2600 ext. 1642
fax: +1 (613) 520-4334
weiss@scs.carleton.ca
http://www.scs.carleton.ca/~weiss/

Daniel Amyot

SITE, University of Ottawa
800 King Edward
Ottawa, ON, K1N 6N5 (Canada)

phone: +1 (613) 562-5800 ext 6947
fax: +1 (613) 562-5664
damyot@site.uottawa.ca
http://www.site.uottawa.ca/~damyot/

Abstract. This article demonstrates how the User Requirements Notation (URN) can be used to model business processes. URN combines goals and scenarios in order to help capture and reason about user requirements prior to detailed design. In terms of application areas, this emerging standard targets reactive systems in general, with a particular focus on telecommunications systems and services. This article argues that the URN can also be applied to business process modeling. To this end, it illustrates the notation, its use, and its benefits with a supply chain management case study. It then briefly compares this approach to related modeling approaches, namely, use case-driven design, service-oriented architecture analysis, and conceptual value modeling. The authors hope that a URN-based approach will provide usable and useful tools to assist researchers and practitioners with the modeling, analysis, integration, and evolution of existing and emerging business processes.

Keywords. Business process modeling, goal-oriented analysis, scenario modeling, service-oriented architecture, use cases, User Requirements Notation

INTRODUCTION

Business process modeling (BPM) is a structured method for describing and analyzing opportunities of improving the business objectives of stakeholders, including providers and customers. BPM usually involves identifying the roles of users involved in the process, and the definition of activities (often described as workflows or services) that contribute to the satisfaction of well-defined business goals. Approaches for BPM are business-centric rather than technology-centric, although connections to designs and implementations (for example, via mappings to workflow engines or Web services) are also desirable.

BPM approaches need to address the well-known "W5 questions": *Why* do this activity? *What* should this activity be precisely? *Who* is involved in this activity? *Where* and *when* should this activity be performed? Additionally, a business process model should enable ways of (formally) analyzing the processes and goal satisfaction. Finally, business process models should be understandable to various stakeholders, including customers.

Several years ago, the standardization sector of the International Telecommunications Union initiated work towards the creation of a *User Requirements Notation* (URN) in the Z.150 series of Recommendations (ITU-T, 2003). The purpose of URN is to support, prior to

detailed design, the modeling and analysis of user requirements in the form of goals and scenarios, in a formal way. URN is generally suitable for describing most types of reactive and distributed systems, with a particular focus on telecommunications systems and services. The applications range from goal modeling and requirements description to high-level design. An overview of URN with a tutorial example from the wireless communication domain is presented in (Amyot, 2003). Annex A also includes a summary of the notation.

URN has concepts for the specification of behavior, structure, goals, and non-functional requirements, which are all relevant for business process modeling. URN is in fact composed of two complementary notations, which build on previous work. The first one is GRL, the *Goal-oriented Requirement Language* (URN Focus Group, 2003a). For the last decade, goal-oriented modeling has been a very active field in the requirements engineering community (Yu and Mylopoulos, 1998). One well-established language is the NFR (Non-Functional Requirements) framework, published in (Chung *et al.*, 2000). GRL includes some of the most interesting concepts found in the NFR framework and complements them with agent modeling concepts from the *i** framework (Yu, 1997). GRL captures business or system goals, alternative means of achieving goals, and the rationale for goals and alternatives. The notation is applicable to non-functional as well as functional requirements.

The second part of URN is the *Use Case Map* (UCM) notation, described in (URN Focus Group, 2003b). The UCM notation was first defined by Buhr and his colleagues (Buhr and Casselman, 1996; Buhr, 1998) to depict emerging behavioral scenarios during the high-level design of distributed object-oriented reactive systems. It was later considered appropriate as a notation for describing operational requirements and services. A UCM model depicts scenarios as causal flows of responsibilities that can be superimposed on underlying structures of components. UCM *responsibilities* are scenario activities representing something to be performed (operation, action, task, function, etc.). Responsibilities can potentially be allocated to *components*, which are generic enough to represent software entities (e.g., objects, processes, databases, or servers) as well as non-software entities (e.g., actors or hardware resources).

Through an illustrative example, we will argue that URN presents suitable and useful features for modeling and analyzing business processes, and that it satisfies the goals of a BPM language. Our example is based on a WS-I (Web Services Interoperability) case study (WS-I, 2003a). This document describes usage scenarios defining the use of Web services in structured interactions and identifying basic interoperability requirements. It is sufficiently rich in order to exercise the various features of URN, but, at the same time, it is a simplified model of a supply chain management system which can be understood in its entirety.

In this article, we first give an overview of the supply chain management case study as well as of the corresponding UCM model we constructed. Then, we discuss how URN models can be used to analyze architectural changes. Service provisioning relationships for mapping the business process model to Web services are then explored, before looking at paths to detailed service design and validation. We finally discuss related work and present our conclusions.

SUPPLY CHAIN MANAGEMENT: OVERVIEW AND UCM MODEL

In this section we describe how a UCM model can be constructed based on given use cases. We first give an overview of the high-level requirements, followed by a subsection for each use case. It should be noted that we are not mapping each use case to a separate map in the UCM model. Instead, we create a single, so-called *root* map that incorporates the other maps through a hierarchical abstraction mechanism. The UCM model presented here was created with the UCMNAV tool (UCM User Group, 2003).

Overview of High-Level Requirements

The WS-I case study (WS-I, 2003a) provides a high-level definition of a supply chain management system for consumer electronic goods. The requirements are specified in the form of a use case model integrating high-level functional requirements, a set of simplifying assumptions, and a set of use cases and activity diagrams. Non-functional requirements of the nature considered by URN are not specified.

There are five high-level functional requirements in this system:

— Retailer offers consumer electronic goods to Consumers.
— Retailer needs to manage stock levels in Warehouses.
— Retailer must restock a good from the respective Manufacturer's inventory, if its stock level falls below a certain threshold.
— Manufacturer must execute a production run to build the finished goods, if a good is not in stock. (Ordering from suppliers is not modeled).
— Use cases contain logging calls to a MonitoringSystem to monitor services from a single monitoring service.

These requirements already explicitly specify a set of five actors (in sans serif). We therefore take these actors as given, although we can still reason about whether some of those actors can be made internal actors (e.g., the warehouses could be considered a part of the retailer). However, in a typical application of URN, one of the tasks would be to identify this set of actors from informal requirements or from the UCM model, i.e., by considering how the responsibilities we have discovered should be allocated to components.

This system is interesting because it incorporates features of B2C (e.g., between retailer and the consumers) and B2B (e.g., between the retailer and the manufacturer). These two business models also imply different communication needs (e.g., asynchronous communication in B2B vs. typically synchronous communication in B2C).

The WS-I case study specifies eight use cases, and we will map six of them to URN: #1) *Purchase Goods*, #2) *Source Goods*, #3) *Replenish Stock*, #4) *Supply Finished Goods*, #5) *Manufacture Finished Goods*, and #7) *Log Events*. The *Run and Configure Demo* use case (#6, not mapped) addresses one of the goals of the WS-I case study, namely, to demonstrate the interoperability of different vendors' Web services implementations. However, our objective here is to model the business process in a representative supply chain management system, and the demonstration aspects are outside of our scope. The *View Events* use case (#8, not mapped) describes a management functionality that has been removed for space reason.

Use Case 1: Purchase Goods

This use case gives a high-level overview of the business process as a whole, which includes submitting and fulfilling orders. This corresponds to the root UCM shown in Fig. 1. A consumer visiting the retailer Web site expresses her intent to purchase goods by submitting an order. The retailer system replies by fulfilling the order. There are two possible outcomes: RejectOrder, and ShipmentConfirmed. The [NoSuchProductOrCannotBeShipped] path is taken if any of the products in the order do not exist (in this case the whole order is rejected), or none of the items can be shipped. In the [OrderSuccessful] path, a shipping confirmation is returned with a list of items shipped, indicating the quantity shipped for each.

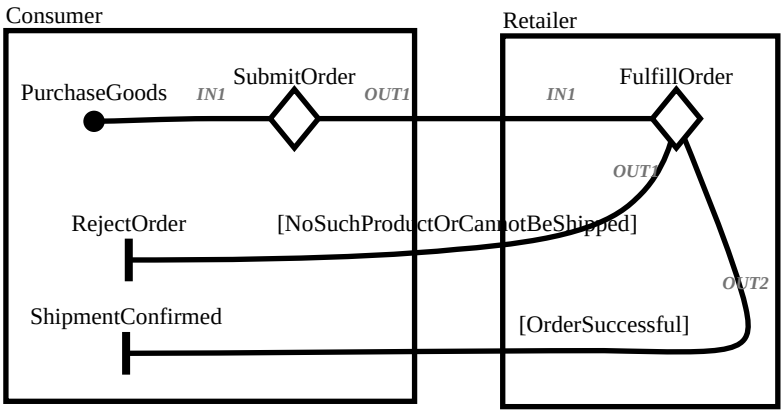


Fig. 1. BusinessProcess root map.

In the UCM notation (see Fig. 19), scenarios are initiated at *start points*, represented as filled circles, and terminate at *end points*, shown as bars. *Paths* show the causal relationships between start and end points. Generic *components* are shown as rectangles, and they are responsible for the various activities (called *responsibilities* and indicated by X's on a path) allocated to them. Labels for *guarding conditions* are shown between square brackets. Diamonds are used to represent *stubs*, which are containers for submaps called *plug-ins*. Stubs have named input and output segments (e.g., *IN1* and *OUT1* in Fig. 1) that are bound to start and end points in a plug-in, hence ensuring the continuation of a scenario from a parent map to a submap, and to the parent map again.

The BusinessProcess root map contains two stubs, each of which with one submap: SubmitOrder and FulfillOrder. In SubmitOrder the consumer navigates to the shopping site, and the system responds with the product catalog. The consumer then enters the order information and submits the order. This submap is shown in Fig. 2. In FulfillOrder, shown in Fig. 3, the retailer checks with its warehouses whether they can supply the items in the order (assuming the requested product exists), and asks them to ship the items. This use case incorporates the *Source Goods* use case, described in the next section.

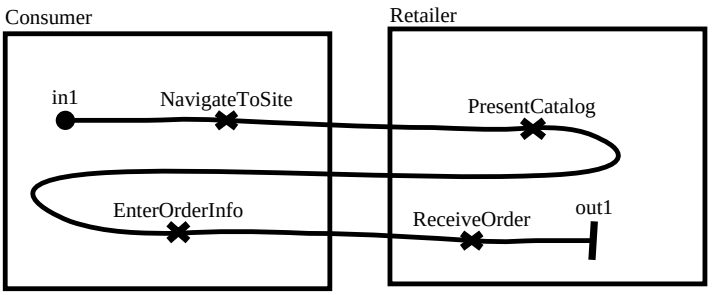


Fig. 2. SubmitOrder submap.

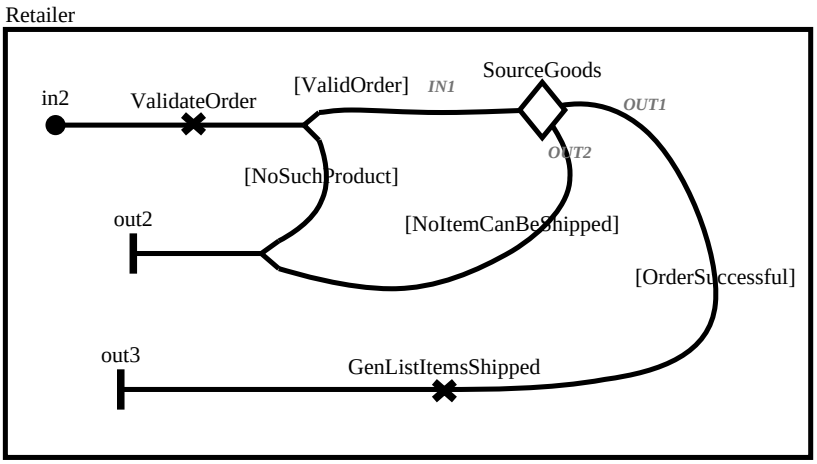


Fig. 3. FulfillOrder submap.

Use Case 2: Source Goods

In this use case, the retailer tries to locate the ordered goods in its warehouses. If the requested quantity of a given item is available, the retailer requests its shipment. Otherwise, it will record that the item could not be shipped. (As stated in the requirements, requests can only be fulfilled in full. Stocks from multiple warehouses cannot be combined.) The use case results in a list of the line items that each warehouse will ship, and accordingly adjusted inventory levels.

The submap corresponding to this use case is shown in Fig. 4. It is important to note that in the UCM model we do not need to map each use case separately, but we can integrate several of them in the same diagram. The complexity of the resulting model can be reduced through hierarchical abstraction, as provided by stubs and plug-ins.

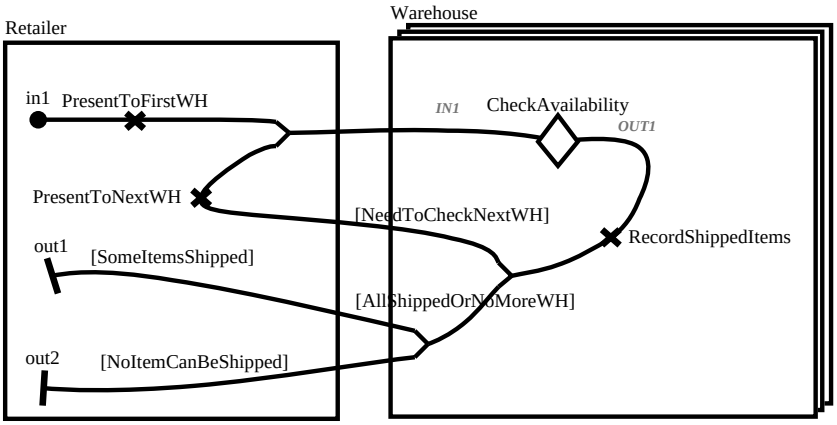


Fig. 4. SourceGoods submap.

The CheckAvailability submap in Fig. 5 shows the iteration through the list of items presented to an individual warehouse. Whenever an item is available, the ordered quantity is decremented from the warehouse inventory.

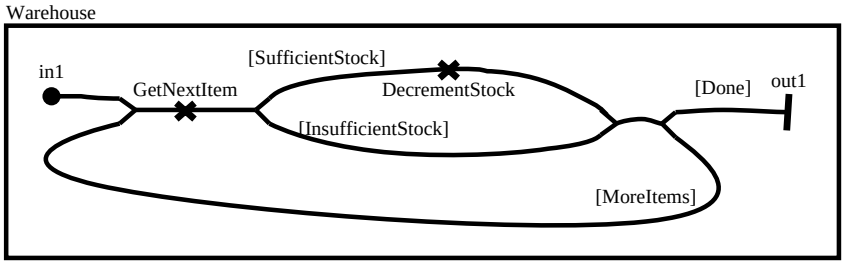


Fig. 5. CheckAvailability submap.

In the next section, we will "attach" the UCM for the *Replenish Stock* use case (discussed in the next subsection) to this submap in order to express that stock replenishment is triggered asynchronously whenever the stock level for a particular item gets below a given threshold after decrementing the stock. We will also discuss an alternative approach and compare both approaches using architectural change analysis.

Use Case 3: Replenish Stock

The warehouse orders goods from manufacturers to replenish its own stock for a given product. The submap for this use case is shown in Fig. 6. This map is interesting as it demonstrates the use of parallelism with a UCM *AND-fork*. Upon receiving and validating the order, the selected manufacturer immediately acknowledges the receipt of the order before it starts processing the request (first of two parallel branches, which ends in AckToWH). The reason for this is that the manufacturer may need to produce the requested goods before it can supply them, if it has insufficient inventory of the product (second parallel branch).

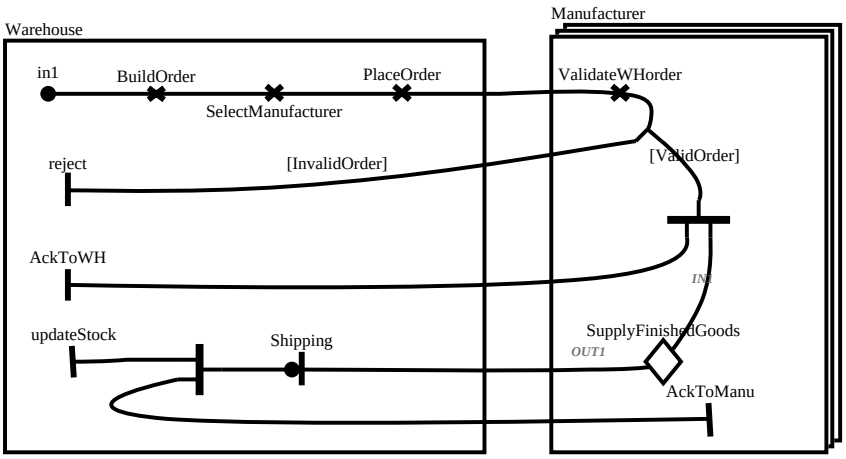


Fig. 6. ReplenishStock submap.

As soon as the manufacturer has shipped the finished product, it sends a shipping notice to the warehouse (Shipping). In response the warehouse updates its inventory, and acknowledges the receipt of the shipping notice to the manufacturer (AckToManu). Again, an AND-fork is used to indicate that these responsibilities are performed in parallel.

Use Case 4: Supply Finished Goods

The manufacturer receives a purchase order from a warehouse. The manufacturer may either be able to satisfy the request with the inventory at hand, or may need to manufacture the requested goods. (As stated, these orders only contain a single line item.) The submap for this use case is shown in Fig. 7. It makes use of a dynamic stub to represent the optional step of

manufacturing finished goods, with two plug-ins as detailed in the submaps for ManufactureFinishedGoods (Fig. 8 and Fig. 9).

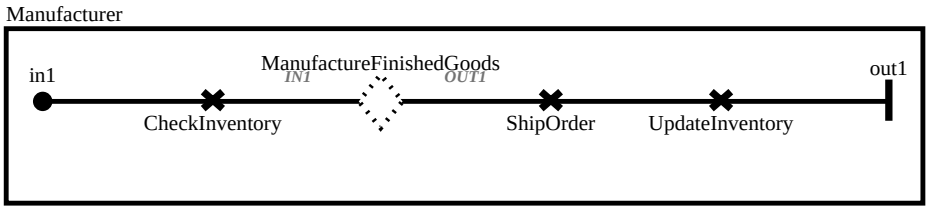


Fig. 7. SupplyFinishedGoods submap.

From the context (the ReplenishStock submap), it is clear that we do not need to validate the order again at the beginning of this submap. However, if SupplyFinishedGoods was turned into a service at a later stage, we would also need to check the input, and reject purchase orders as necessary. As a general principle, services should, therefore, not make any assumptions about the context in which they will be invoked, e.g., whether their input is correct.

Use Case 5: Manufacture Finished Goods

For this use case, two plug-ins (used in the stub of Fig. 7) are required. The first one is a simple pass through, and is selected when the manufacturer holds sufficient inventory of the product (Fig. 8). With no component specified, this plug-in can be reused more easily in other stubs. The second plug-in describes how the finished goods are produced when the inventory is insufficient. First, the required parts and quantities are determined, then the good is assembled, and the manufacturer's inventory is updated (Fig. 9). The dynamic stub defines the condition under which each plug-in will be selected (also known as selection policy). These conditions will be formalized in the subsection presenting scenario definitions.

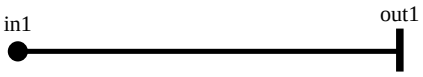


Fig. 8. Default submap.

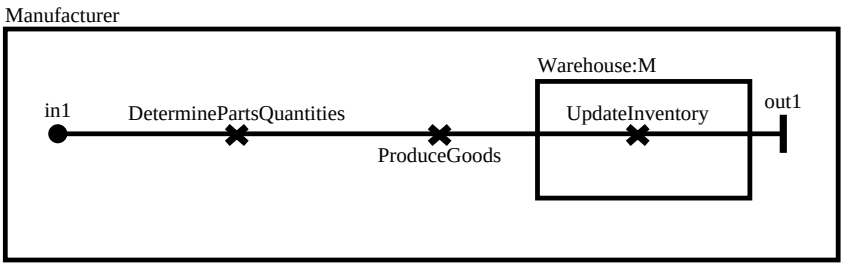


Fig. 9. ManufactureFinishedGoods submap.

Note that the manufacturer has its own warehouse where it stores its inventory. The inventory is considered insufficient when its level falls below a minimum threshold, or the quantity ordered is larger than the current level. A manufacturer is assumed to be able to produce any items requested, but not beyond a maximum inventory level.

Use Case 7: Log Events

This use case is concerned with logging events relating to the execution of the other use cases. Events can be logged for any number of reasons, including debugging, maintenance, or non-repudiation. The corresponding UCM in found in Fig. 10. One shortcoming of any

scenario-based notation is that logging the execution of a scenario is difficult to model. Of course, one could insert a LogEvents stub after each responsibility that we want to log, but that would make the diagrams clumsy. Logging is best modeled as an *aspect* of a responsibility that is executed whenever the responsibility is performed. Other examples of aspects are encryption (e.g., to model that messages are sent in encrypted form), and billing.

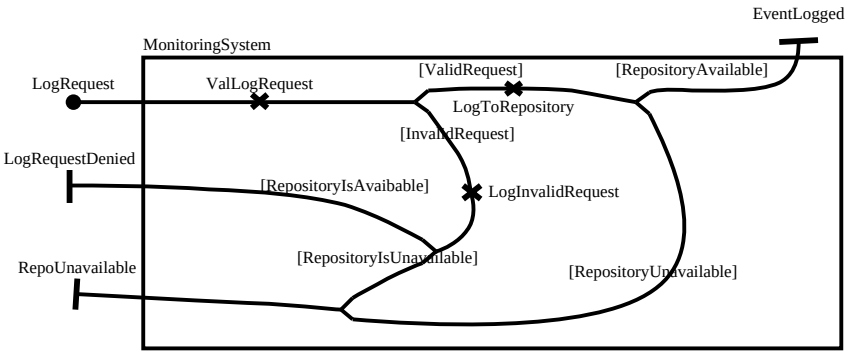


Fig. 10. LogEvents submap.

When Fig. 10 shows a submap, it needs to be interpreted as a plug-in that is (logically) inserted after each responsibility that we want to log in any of the other maps (LogRequest would be triggered in passing). In this use case, the event may actually not be logged, if the request is invalid or if the repository is unavailable.

ARCHITECTURAL CHANGE ANALYSIS

One of the benefits of URN is the ability to reason about architectural changes. In this section, we will look at several examples where URN allows us to consider different types of trade-offs: functional, non-functional, and structural.

Triggered vs. Periodic Replenishment of Stock

In the presentation of the UCMs, we have not discussed under which circumstances the ReplenishStock scenario (Fig. 6) is executed. We consider two architectural alternatives: the scenario can be triggered as a result of decrementing the stock level in the CheckAvailability submap (Fig. 5), or it can be executed periodically. This is a functional trade-off, since the alternatives realize two functionally different ways of implementing the same requirement.

To implement the first alternative, we can extend the CheckAvailability submap by "touching" it with a path leading to a Replenishment stub. This means that the path is to be executed asynchronously (i.e., in parallel) once the DecrementStock responsibility has been performed. By making Replenishment a dynamic stub, we can specify a selection policy to decide whether the stock needs to be replenished or not. The necessary changes are indicated in Fig. 11.

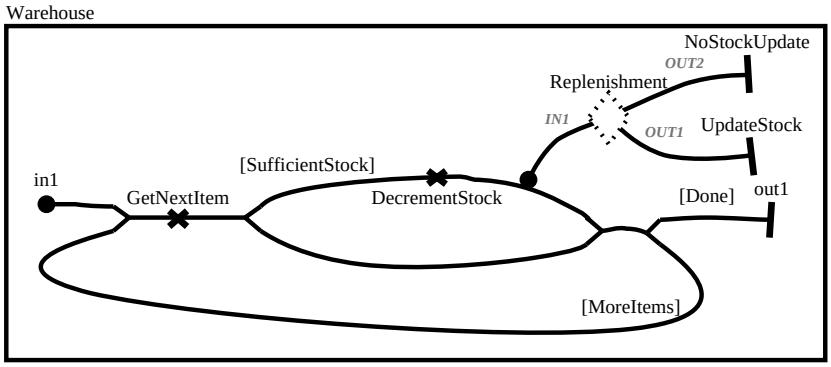


Fig. 11. Alternative 1: Triggered replenishment.

The architectural alternative to triggered replenishment is to replenish the stock periodically. This does not require any changes to CheckAvailability, and can be accomplished by a new root map for periodic replenishment shown in Fig. 12. This map contains a timer, represented with a clock symbol. ReplenishTimer is reset when the StopPeriodicRep scenario reaches it in time, otherwise this timer expires and the time-out path (squiggly line) is taken.

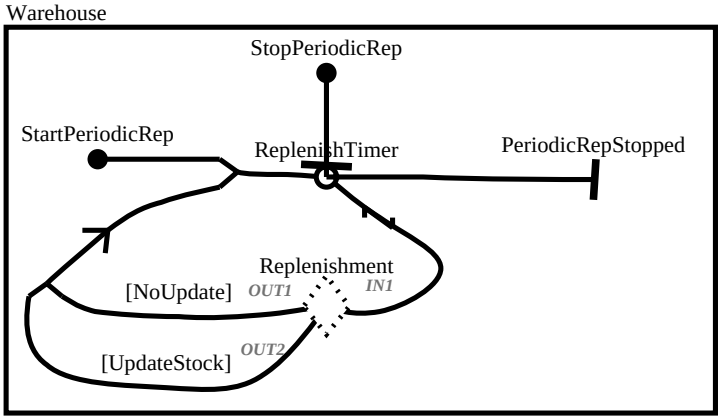


Fig. 12. Alternative 2: Periodic replenishment.

The choice between these alternatives involves a trade-off between availability and maintainability and manageability. The GRL model in Fig. 13 provides details of the trade-off. It represents the two architectural alternatives as *tasks* (hexagons), and the three non-functional requirements as *softgoals* (clouds) to be achieved. In GRL, model elements can contribute to each other, and this is shown with arrows. Contributions can be positive or negative, as well as sufficient or insufficient (see the legend in Fig. 18).

Our definitions of the non-functional requirements used in this model and subsequent ones are provided in Table 1.

Table 1. Definition of non-functional requirements in the BPM context.

NFR	**Definition**
Availability	Ability to handle requests (e.g., in terms of the number of fulfilled orders)
Maintainability	Ability to evolve the system, and to fix errors
Manageability	Ability to monitor system performance and adjust parameters
Performance	Speed of performing business functions
Scalability	Ability to grow the system (e.g., number of users)
Security	Identification, secrecy, integrity, access control, and audit

The model in Fig. 13 states that, using triggered replenishment, a warehouse can detect more quickly that inventory levels are getting low than when using periodic replenishment.

Consequently, we will be able to keep up with higher than anticipated demand (up to the limit of the manufacturer's capacity to produce, of course). When using periodic replenishment, we check inventory levels only when triggered by a timer, and the length of the period affects how soon we can respond to changes in demand beyond what we had anticipated when setting the length of the period.

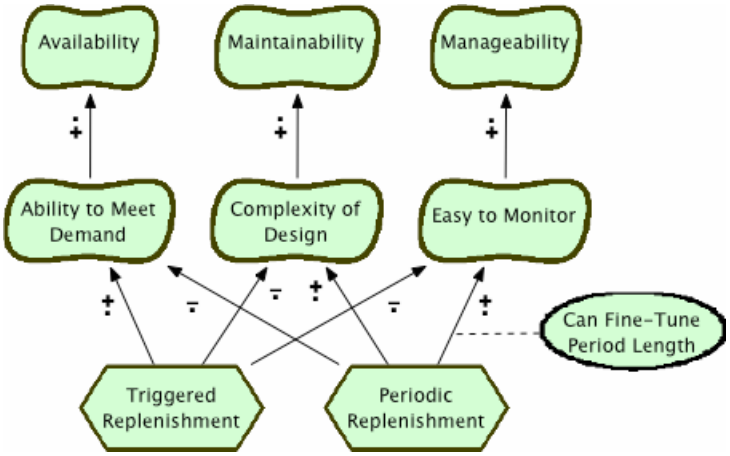


Fig. 13. GRL model to compare between triggered and periodic replenishment.

On the other hand, the first alternative leads to a more complex design, because we need to distribute the checks for inventory levels wherever the level can change. In the second alternative, the checking logic is centralized, and we do not need know at what points in the business process the inventory level changes. Although setting the length of the period appropriately makes the configuration of the system slightly more difficult, this disadvantage is more than offset by the greater ease with which the system can be managed in the second alternative, which allows us to monitor the inventory levels, and adjust the length of the period as necessary.

The reasons we just provided for giving a contribution link a particular weight can also be expressed directly in the diagram in the form of *beliefs*, shown as ellipses. For example, we have justified why Periodic Replenishment makes inventory levels Easy to Monitor in Fig. 13. Another stakeholder could have a different opinion on this contribution, and her belief could also be added to the model, hence documenting arguments and rationales until agreement is reached. It is often a judgment call on what level of detail to present in the diagram, as its main reason is to summarize the reasoning succinctly, and to enable trade-offs in a given context. We therefore do not want to overload the diagram with additional annotations.

Note that the decision regarding triggered or periodic replenishment is not done in the absolute for either one of the alternatives. Which alternative is chosen depends on the context, i.e., the priorities of the user (which NFRs are most important). For instance, if our goal was a more maintainable design, periodic replenishment would be the preferred solution, albeit at lower availability. Diagrammatically, we can include the context in the GRL model via a goal such as "*total satisfaction*", and indicate the importance of each existing top-level softgoal through an appropriate contribution link (e.g., make or some-).

Local Logging vs. Using a Centralized Logging Service

The requirements specified the use of a centralized logging service. However, whether to log locally or centrally is often a (non-functional) trade-off faced by an architect. There are benefits and liabilities to either architecture, and they are summarized in Fig. 14. In this case, a trade-off is made between performance, on one hand, and manageability and maintainability,

on the other. Logging, independently of how it is implemented, improves traceability, and, therefore, maintainability. However, local logging results in multiple logs, and filtering those logs for errors or unusual behavior will be harder than if a central logging service is used. Thus, on one hand, central logging leads to a more manageable system. On the other hand, however, central logging will lower performance because of the network overhead.

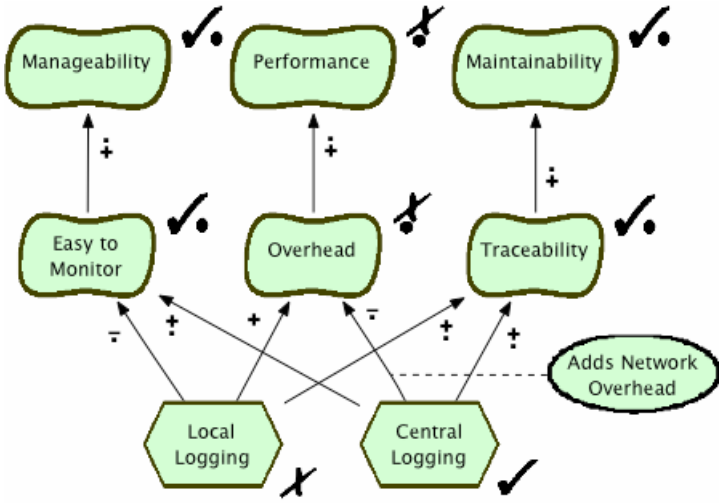


Fig. 14. GRL model to compare between local and central logging.

In addition to the contributions of each architectural alternative, this diagram also shows the results of propagating the effect of choosing one of the alternatives. Evaluations of GRL graphs show the impact of qualitative decisions on high level softgoals. A tool like OME (Yu and Liu, 2000) can automatically propagate the labels assigned to leaf nodes of the graph. The GRL propagation algorithm discussed in (URN User Group, 2003a) and supported by OME is inspired from (Chung *et al*. 2000). Propagation is usually bottom-up and takes into consideration three parameters, whose notations are presented in Fig. 18:

— Contributions and correlations (positive or negative, sufficient or not)
— Degrees of satisfaction (satisfied or denied, weakly or fully)
— Composition operators (AND, OR)

Evaluating GRL models usually provides more complete answers than using simple benefits/drawbacks tables or criteria evaluation matrices. One could also use numerical values and functions instead of qualitative (fuzzy) values, although the latter one are often more appropriate in the early development stages.

As we can see in Fig. 14, choosing central logging positively affects manageability, and maintainability (weakly satisfied), and negatively affects performance (weakly denied). A similar model for the other alternative would lead to opposite results. In general, many combinations of tasks and softgoals can be evaluated, and the results are not always so clear.

Modeling Warehouses as Internal Components

Our final example illustrates a structural trade-off. In addition to allocating responsibilities to UCM components in different ways, we can also nest components within other components. Such nesting then implies that those components are tightly coupled and that the nested component is not visible to top-level components. An example of this in the supply chain management case study is how we model warehouses.

Both alternatives have already been illustrated previously. While the submap in Fig. 4 models the retailer's warehouses as top-level components, the manufacturer's warehouse is modeled as a nested component in Fig. 9. The trade-offs between these alternatives are sum-

marized in Fig. 15. As shown, modeling warehouses as top-level actors trades off maintainability, scalability, and availability for performance and manageability.

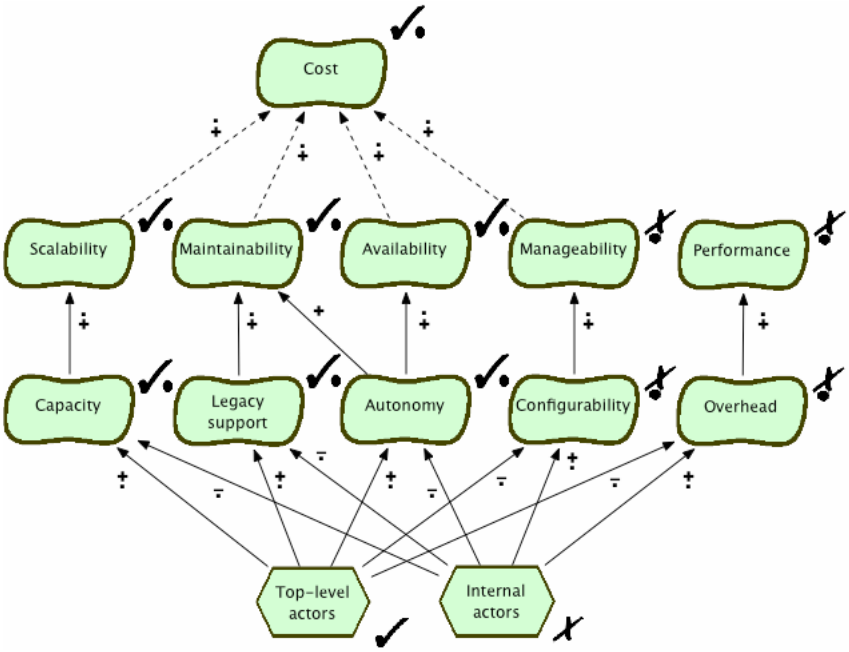


Fig. 15. GRL model to compare between top-level and internal actors

This alternative improves maintainability, because of its support for legacy solutions (e.g., warehouses developed within different business units), and autonomy, as it allows for the warehouse service to be provided by a third party (e.g., FedEx). If internalized, the warehouses would need to adopt a common operating platform and/or database schema. It increases scalability as it allows the system's capacity to grow by adding more warehouses. In the other alternative a single retailer database may quickly become a bottleneck.

Availability is higher when representing warehouses as independent entities, since there is no single point of failure. If one of the warehouses fails, the others can continue operating. However, since all requests to warehouses will be remote, there is a messaging overhead that reduces performance. Manageability is also decreased since each warehouse needs to be configured separately. As a side-effect (represented with dashed arrows called *correlations*) of improved maintainability, scalability, availability, and at the expense of reduced manageability, the overall (minimized) cost is positively affected.

Such a structural change also has an impact on the organization of the business. When warehouses are treated as top-level components, the functionality of the warehouse could be outsourced to a fulfillment partner such as FedEx or UPS. These companies have evolved the services they provide beyond just fulfillment and delivery services, but can also manage the warehouses of their customers, and even aggregate suppliers' components in transit. The use of URN for reasoning about such organizational issues is further explored in (Weiss and Amyot, 2005).

SERVICE PROVISIONING RELATIONSHIPS

In this section we are concerned with deriving service provisioning relationships between components from the UCM model. The goal of this step is to map the business process model to a Web services architecture. We define a *service* as a collection of related operations implemented by a component. The component that implements a service is known as *service provider*, and components that invoke the service as *service users*. An analysis of the UCM

model provides us with potential operations which can then be grouped into services. Note that our definition of the term service abstracts from the issue of how a service is accessed, that is, network addresses, protocols, and data formats. As we refine service relationships into WSDL Web service definitions, these aspects need to be specified.

Each service operation comprises a set of messages exchanged between a service user and a service provider. In line with the WSDL specification (W3C, 2001) we differentiate between one-way, request-response, solicit-response, and notification operations. Both one-way and notification operations consist of a single message sent, respectively, by the service user, and the service provider. In the case of request-response and solicit-response operations, two correlated messages are exchanged, the difference being who initiates the dialog (the service user in the former, and the service provider in the latter).

In the process of identifying potential service operations we therefore look for causal paths in the UCM model that cut across component boundaries. The path segments consisting of the last responsibility or condition along those paths in the source component, and the first responsibility or condition in the destination component can be identified with a message. Their names can be concatenated to serve as a unique handle for the message. This information can be automatically extracted from a UCM by using the UCMNAV tool. The next section provides further details on extracting messages from a UCM model.

However, whether responsibilities or conditions will actually be exposed as service operations also depends on other considerations. In general, only some of the interactions at the business process level will result in Web service interactions. In our case, since we are working here within the scope of the WS-I case study, our target will be a service-oriented implementation. It should also be pointed out that one important feature of UCM models is that the mapping from a UCM model to an implementation architecture is not one-to-one, and that the same causal dependency between components can be realized using multiple protocols and communication mechanisms, or not map into implementation-level concepts at all. In this section, we are suggesting one way in which UCM models can be mapped to Web service interactions. The process and template described below are intended to provide the foundation for an automatic mapping from UCM models into Web services.

Having identified path segments that represent messages between components, we need to decide which service operations they belong to. At present this decision needs to be made manually, although one could expect to use pattern matching to classify messages into operations given the structure of the UCM model. Subsequently, we need to group those operations into services. As a simple illustration of this process, consider the interactions between consumer and retailer.

From the UCM model, we can extract the following path segments that correspond to messages exchanged between the consumer and retailer components:

c.NavigateToSite-r.PresentCatalog (Navigate)
r.PresentCatalog-c.EnterOrderInfo (Catalog)
c.EnterOrderInfo-r.ReceiveOrder (OrderInfo)
r.OrderSuccessful-c.ShipmentConfirmed (ShipmentConfirmed)
r.NoSuchProduct-c.RejectOrder (RejectOrder)
r.NoItemCanBeShipped-c.RejectOrder (RejectOrder)

To ensure uniqueness, we prefix the name of the responsibility or condition with the corresponding component identifier (c, r, m, or w). In parenthesis we also show the name that is manually assigned to the message during detailed service design (to be discussed in the next section) in order to provide traceability to the MSC model. The message exchange above can be interpreted as two request-response operations, both provided by the retailer component. We can give them more expressive names such as getCatalog() and submitOrder(). Note that submitOrder() has two possible responses: either the order is rejected, or shipment is confirmed.

Our approach suggests grouping related operations into services. In some cases, a component may therefore offer multiple services if they bundle different functionalities. In the example, however, getCatalog() and submitOrder() are both operations related to the ordering process, and we decide to group them into a single Retailer service. In this example, the name of the service is simply derived from the name of the component that provides it.

We propose the following template for documenting the assignment of operations to services. For a given service user and given service provider, we list the messages exchanged, and label them according to their role in an operation (request, response, solicitation, or notification). Finally, we group the messages into operations. The interaction between the consumer and the retailer actors can now be documented as follows:

Consumer-Retailer
Operation: getCatalog()
    Request: c.NavigateToSite-r.PresentCatalog (Navigate)
    Response: r.PresentCatalog-c.EnterOrderInfo (Catalog)
Operation: submitOrder()
    Request: c.EnterOrderInfo-r.ReceiveOrder (OrderInfo)
    Response: r.OrderSuccessful-c.ShipmentConfirmed (ShipmentConfirmed)
    OR Response: r.NoSuchProduct-c.RejectOrder (RejectOrder)
    OR Response: r.NoItemCanBeShipped-c.RejectOrder (RejectOrder)

Similarly, a request-response operation shipGoods() belonging to a Warehouse service can be derived from the interaction between the retailer and its warehouses. However, the interaction between warehouses and manufacturers leads to a different type of operation. The messages exchanged between a warehouse and a manufacturer are as follows:

Warehouse-Manufacturer
Operation: submitPurchaseOrder()
    Request: w.PlaceOrder-m.ValidateWHorder (PlaceOrder)
    Response: m.ValidOrder-w.AckToWH (Ack)
    OR Response: m.InvalidOrder-w.reject (reject)

Manufacturer-Warehouse
Operation: submitShippingNotice()
    Notification: m.UpdateInventory-w.ShippingNotice (ShippedItems)

The first message exchange can be mapped to a request-response operation in the Manufacturer service submitPurchaseOrder(). The reply is in fact a partial acknowledgement of the request, and the actual reply is sent with the next message. The second message exchange corresponds to a notification, resulting in a callback of the Warehouse service. We can, therefore, add a submitShippingNotice() operation to the Warehouse service.

Finally, the interactions with the MonitoringSystem can be exposed as a Logging service. It provides the operations logEvent() and getEvents(). The service provisioning relationships between the components can be visualized as a UML deployment diagram (see Fig. 16). In this diagram we use a convention introduced by (Carlson, 2001) to show both service deployment, as well as service invocations in a way similar to a collaboration diagram.

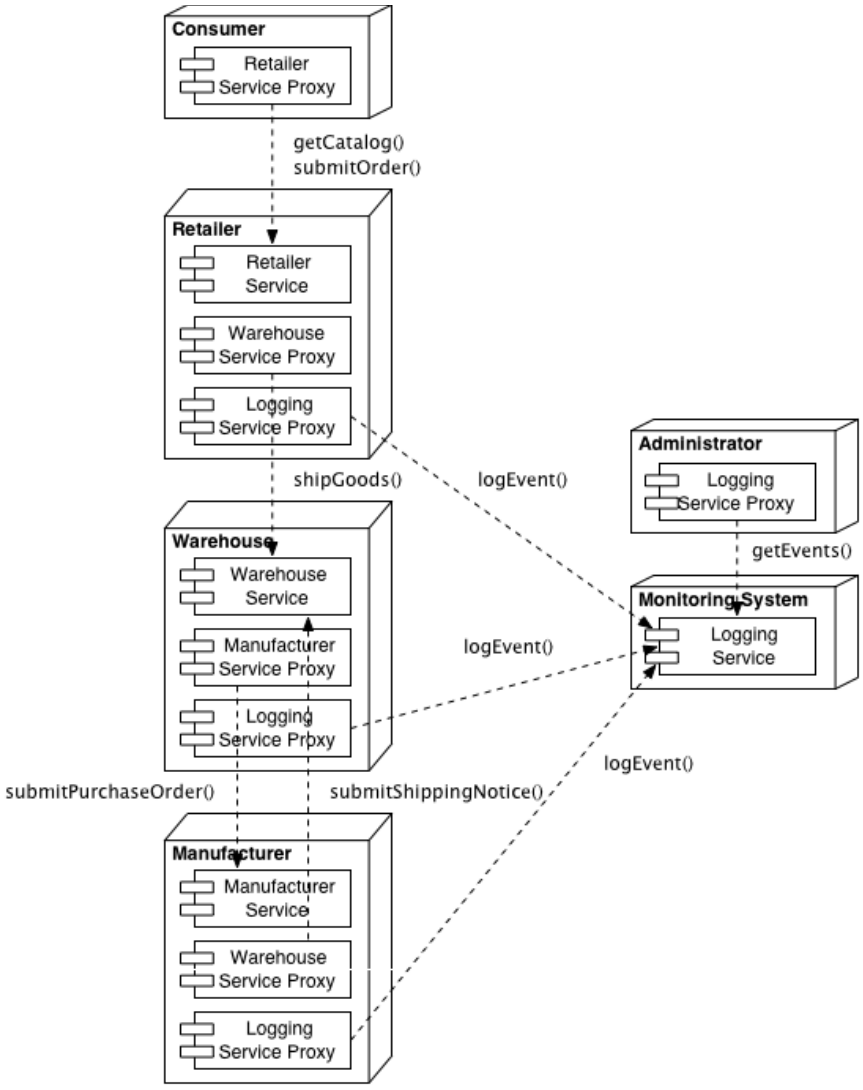


Fig. 16. Service provisioning relationships.

TOWARDS DETAILED SERVICE DESIGN AND VALIDATION

The previous section already motivated the need for messages between components in the context of operations for Web services. This section presents a tool-supported technique, based on scenario definitions and a UCM path traversal algorithm, for automating part of the process of generating messages. This technique has application in the understanding, visualization, simulation, and analysis of UCM models, as well as the generation of design-level scenarios and test goals for validating designs and implementations.

Scenario Definitions

The UCM notation supports a very simple *path data model* that can be used to traverse paths in a deterministic way. Global Boolean control variables can be used to formalize conditions. Responsibilities can also modify the content of these variables with new values resulting from the evaluation of Boolean expressions.

In our case study, several such variables are needed, and Annex B defines them. Using these variables, formal conditions can be attached to branching points in a model, i.e., to OR-

forks (for selecting alternative branches), dynamic stubs (for selection policies), and timers (to decide whether or not a timeout occurs).

For instance, the selection policy of the ManufactureFinishedGoods dynamic stub in Fig. 7 is:

SufficientInventory → Select plug-in Default
¬*SufficientInventory* → Select plug-in ManufactureFinishedGoods

For the OR-fork in the SourceGoods UCM (Fig. 4), the guarding conditions on the two branches are formalized as follows:

[NeedToCheckNextWH] = *WarehouseLeft* ∧ (¬*ItemListEmpty*)
[AllShippedOrNoMoreWH] = (¬*WarehouseLeft*) ∨ *ItemListEmpty*

A UCM model may include several groups of *scenario definitions*. Each such definition consists of initial values for the variables, a set of start points initially triggered, an optional post-condition expected to be satisfied at the end of the execution of the scenario, and a description. Here are two examples (note that *T* means True and *F* means False):

Scenario: **PrimaryScenario**
Description: The warehouse has the desired item.
Starting Point: PurchaseGoods (in map BusinessProcessRoot)
Variable Initializations:
ProductExists = T
SufficientStock = T
WarehouseLeft = F
MoreItems = F
ItemListEmpty = T
StockStillSufficient = T

Scenario: **CheckButStocksSufficient**
Description: The periodic replenishment is checked but stocks are sufficient.
Starting Point: StartPeriodicRep (in map PeriodicReplenishment)
Variable Initializations:
ReplenishTimer_timeout = T
SufficientStock = T

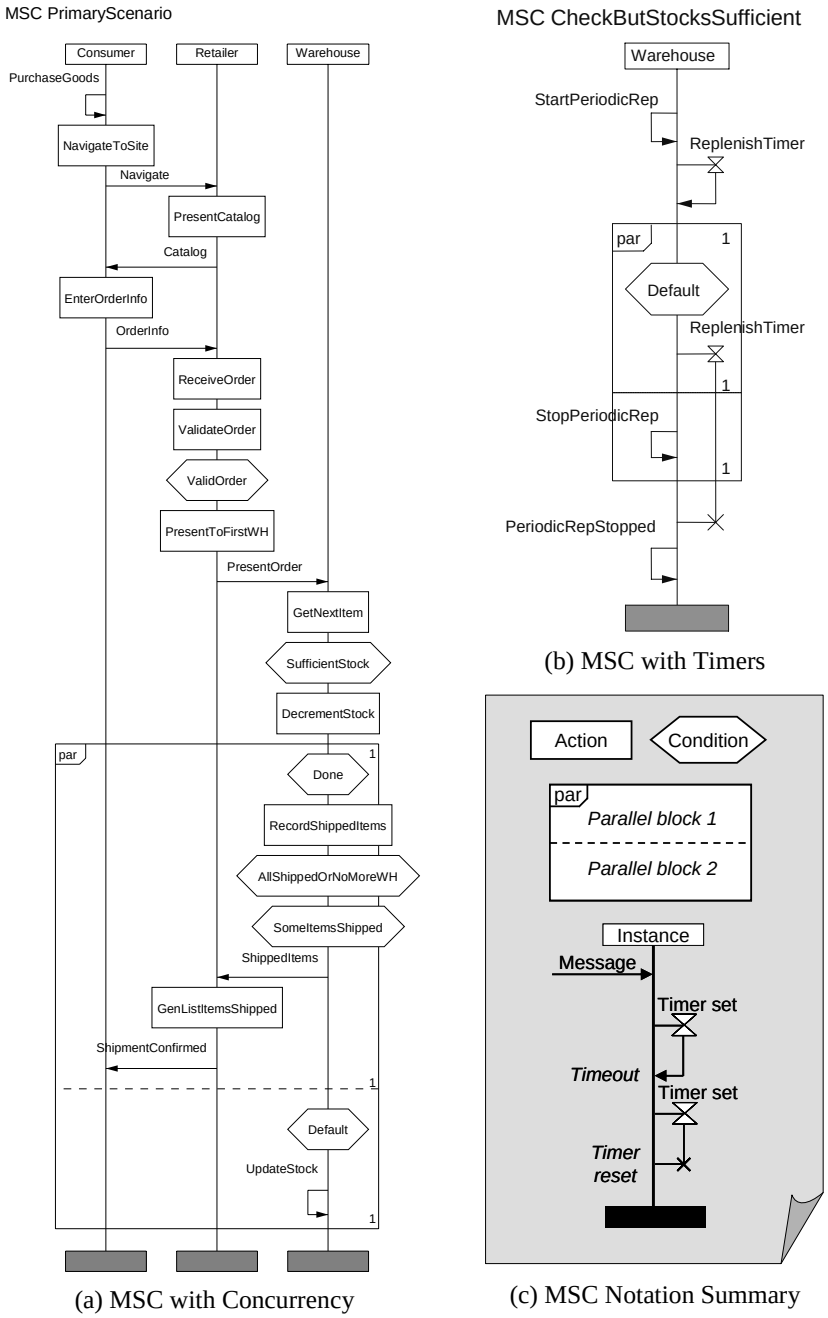
In our model, we created 20 such scenarios definitions categorized in 5 groups. Together, these scenarios cover all the paths from all the root maps and submaps used in our case study. The scenarios resulting from the above two definitions will be illustrated with Message Sequence Charts in the next subsection.

Generation of Message Sequence Charts (MSCs)

The use of scenario definitions for UCM analysis and transformations was pioneered by (Miga *et al*., 2001), who proposed a UCM path traversal analysis that took as input a UCM model and scenario definitions and produced as output a *Message Sequence Chart* (ITU-T, 2004) for each scenario. This functionality was prototyped in UCMNAV. The same mechanism was used to highlight the path traversed in the UCM model according to the scenario definitions, hence helping with the understanding of complex or lengthy individual scenarios. The algorithm was generalized in (URN Focus Group, 2003b), and then re-implemented in UCMNav, this time to generate the output of the traversal in XML (Amyot *et al*., 2003). In a nutshell, the new algorithm uses a depth-first traversal of the graph that captures the UCMs'

structure and generates scenarios where sequences and concurrency are preserved, but where alternatives are resolved using the Boolean variables. If conditions cannot be satisfied or evaluated, then the algorithm reports an error. Another tool, UCMEXPORTER (Amyot, Echihabi, and He, 2004), takes these scenarios in XML and converts them to Message Sequence Charts and UML sequence diagrams (OMG, 2003).

MSCs give a linear view of scenarios that traverse multiple UCMs, a situation that occurs frequently when plug-in maps are used. They are composed of component instances, shown as vertical lines, and of messages, shown as arrows. Fig. 17(c) gives a brief summary of a subset of the notation, which includes actions, conditions, timers, and concurrent behavior.



(a) MSC with Concurrency

(b) MSC with Timers

(c) MSC Notation Summary

Fig. 17. Two more MSC examples, illustrating typical MSC features.

Fig. 17 provides two examples of MSCs automatically generated from the two scenarios defined in the previous section. UCMEXPORTER generates these MSCs in Z.120 textual form, which can then be rendered graphically by tools such as Tau (Telelogic, 2004). To preserve the semantics of UCMs and traceability to the original model, UCM components are shown

as MSC instances, start and end points as messages, condition labels and selected plug-in names as conditions, and responsibilities as actions.

In the first scenario (Fig. 17a), we can observe that a Navigate message shows up in the MSC while it is absent from the UCM model. This message is synthesized automatically by UCMEXPORTER in order to preserve the causal flow between a responsibility in the Consumer component and another responsibility in the Retailer component (see Fig. 2). Message names between pairs of components are provided in a configuration file and can be refined by concrete message exchanges, e.g., in a way consistent with the message names used in the Web service operations defined in the previous section. Consequently, MSCs can be re-generated as the UCM model evolves, or when different architectures are being evaluated.

Fig. 17(a) illustrates the primary scenario for the business process (Fig. 1) with triggered replenishment (Fig. 11). The linear nature of MSCs makes it easy to follow and inspect this scenario, which otherwise would require the stakeholders to flip back and forth through six different UCMs in order to get the same understanding. This scenario is also interesting because it preserves the concurrency specified at the UCM level (e.g., with AND-forks). The MSC in Fig. 17(b) describes a situation where a timer is used (shown in the PeriodicReplenishment submap in Fig. 12). The ReplenishTimer is set when first traversed, times out, is set again in a second iteration of the looping UCM path, and is finally reset when StopPeriodicRep occurs.

The MSC notation is more interesting than UML 1.5 sequence diagrams because of its support for concurrency and timers. However, the new UML 2.0 sequence diagrams, being based on the MSC standard, now support the same concepts. Hence, generating UML 2.0 sequence diagrams from UCMs is a valid alternative to generating MSCs.

The generation of detailed scenarios from higher-level UCM descriptions respects the spirit of model-driven development. In OMG's Model Driven Architecture (OMG, 2004), platform-independent models are refined into models containing platform-specific information. Indeed, as seen in this section, platform-specific communication information can be added to the scenarios generated from a UCM model.

It should also be noted that the MSCs generated here are comparable in content to the sequence diagrams found in (WS-I 2003b), with the addition of concurrency and timer information. The MSCs are also defined at the same level of granularity and abstraction, they are traceable to the UCM model, and they are consistent with each other. These aspects cannot be taken for granted when sequence diagrams are created manually.

Application to Validation

Scenario definitions based on path control variables, together with path traversal algorithms, and transformations to other formalisms, contribute greatly to many validation activities:

— The UCM model itself can be "simulated". For instance, this section presented several scenario definitions (based on variables described in Appendix B) used to generate end-to-end scenarios, that is, executions of the model in a given context. Scenario definitions can be seen at test cases that can be used to ensure nothing is broken as the UCM services or architecture evolve, in a way somewhat compatible with the test-development approach proposed by the *agile* development community (Beck, 2003). Errors are reported when the traversal stops (because of non-determinism, unsatisfied conditions, or a start point that is not triggered) or when scenario post-conditions are not met.

— Different stakeholders can review, inspect, and validate individual MSC scenarios extracted from a UCM model, in order to create a shared understanding or reach agreement on issues identified in GRL models. MSCs provide a projection of a UCM model for a specific context (a scenario definition) whose existence can be motivated by a GRL model (Amyot, 2003). Additionally, an MSC provides a view to the UCM model that

may be more familiar to stakeholders that are closer to the implementation of the system or that have previous knowledge of UML sequence diagrams.

— Test goals can be generated from these scenarios. For instance, Amyot, Weiss, and Logrippo (2004) illustrate the benefits of using UCMNav to automate the generation of test goals from UCM models when compared to approaches based on testing patterns or transformations to formal executable specification languages. These test goals, whether represented as MSCs, in XML, or with some other format, can be refined into concrete test cases for the implementation or for the design, if specified in an executable formalism. The traceable correspondence between the names of the messages used in the MSCs of the previous subsection and those used to implement the operation of the Web services improves the reusability of the scenarios in a testing context, e.g., for Web service testing.

— Performance annotations can be added to UCM models in order to generate analyzable performance models in languages such as Layered Queueing Networks. Such a transformation was automated and integrated to UCMNAV by (Petriu *et al.*, 2003). This approach has been successfully applied to several examples from on-line bookstores to video servers and telephony systems (Petriu and Woodside, 2002). Performance tradeoffs that are difficult to resolve qualitatively at the level of a GRL model can hence benefit from quantitative results (e.g., throughput, bottlenecks, service times and demands, resource utilization) generated by these performance models. In our case study, although we have not done it here, one could derive performance models to determine the best value for the replenishment period, or the messaging overhead of central logging, that is, models that allow us to substantiate the existing models (GRL, UCM) with greater level of detail and precision.

RELATED WORK

This section discusses related work, with a particular emphasis on use case-driven design, service-oriented architectures, and conceptual value modeling.

Use-Case Driven Design

Use cases can be supplemented or, to some extent, replaced by URN models. The use case approach has a number of well-known disadvantages that can be averted by using UCMs to model the early requirements of a business process. Using *extends* and *includes* relationships if often difficult, whereas the same functionality is achievable in a simpler way with UCM stubs and plug-ins. UCM models provide a more systematic way of modeling concurrent behavior, and analyzing the interaction of multiple scenarios. Use-case driven approaches rarely provide notions of modeling design goals and linking them to other design artifacts.

The modeling exercise described in this article using URN to describe the WS-I example uncovered a number of inconsistencies, and redundancies in the use case model (WS-I 2003a). These are due to a number of factors, but in particular to the lack of support for showing scenarios belonging to different use cases in the same model, and a lack of an abstraction mechanism similar to that provided by stubs and plug-ins. Activity diagrams, which lack the equivalent of dynamic stubs, 2D component layouts, and scenario definitions, tend to be less readable and harder to use than UCMs for documenting individual scenarios.

Service-Oriented Architecture

The Service-Oriented Architecture (SOA) approach proposed by (Endrei *et al.*, 2004) aims to align services with business goals. In this approach, services are large-grained activities at

the use-case level that a business exposes to be incorporated into other business processes. SOA aims to support many design activities, including domain decomposition, goal-service model creation, and subsystem analysis. During domain decomposition a value chain model of the business domain showing the main functional areas and their interaction is created. The functional areas are further decomposed into business processes, and business use cases.

During goal-service model creation, high-level business goals are decomposed into lower-level subgoals that can be realized by services. This is done to ensure traceability between business needs and implementation using services. During subsystem analysis, business use cases are refined into system use cases, which are then associated with subsystems. Components within each subsystem realize system use cases. For example, a business use case *Purchase Goods* could be refined into the system use cases *Get Catalog* and *Submit Order*. Both would be realized by a RetailerService component.

URN provides support for these three parts, as well as better guidance on what should do be done at each step, and better modeling of scenario interactions.

Conceptual Value Modeling

Conceptual value modeling or *e*^{3}*-value* (Gordijn, 2002; Gordijn and Akkermanns, 2003) is an approach for precisely describing and evaluating innovative e-business ideas. It provides means to evaluate the feasibility of an e-business model focusing on the creation, exchange, and consumption of objects (i.e., the revenue streams) in a multi-actor network. Value models are different from business process models in that the former show how objects of economic value are created and handled by actors, whereas the latter focus on how exchanges of value objects are put into operation from a business process perspective.

e^3*-value* is similar to our approach in terms of its use of scenarios to model causal flows. It also provides a means for performing value-based trade-offs. However, unlike in URN, value is mainly expressed in monetary terms; other non-functional goals cannot be modeled directly, but need to be mapped to how they generate revenues for an actor. On one hand, *e*^{3}*-value* is much more specific in scope than URN. On the other hand, we can think of *e*^{3}*-value* as an intermediary view between general GRL models and operational UCM models. Both approaches could therefore be integrated for modeling e-commerce systems.

Other Related Approaches

The User Requirements Notation and similar languages have been exploited in various contexts, some of which are related to the one presented here:

— UCMs are used by (de Bruin and van Vliet, 2001) for describing and selecting appropriate architectures. An architecture generator produces a candidate software architecture based on feature-solution graphs (which could be expressed to some extent in GRL) connecting quality requirements and solutions (expressed as potential UCM plug-ins for a reference architecture). The architecture is then evaluated, mainly by inspection, against functional and non-functional requirements.

— Both UCM and GRL are used by (Liu and Yu, 2003) to model information systems in a social context specified in terms of dependency relationships among agents and roles. Their approach includes an iterative process where the use of GRL and UCM is intertwined: scenarios refine solutions to goals (*tasks*), and the elaboration of scenarios can lead to the discovery of new goals. It is illustrated with a Web-based training system.

— Many GRL concepts are formalized in the TROPOS agent-oriented methodology. (Lau and Mylopoulos, 2004) explore the use of TROPOS in the context of Web service design, with a greater emphasis on actors and their dependencies (also supported by GRL) than what was presented here. They provide a Customer Relationship Management system example.

It is suggested to use the Agent-based Unified Modeling Language to refine the goal and actor models, especially with activity and sequence diagrams, prior to defining Web services in WSDL. However, this step is not illustrated at all in their example.

— (Bleistein et al., 2004) use GRL to link requirements for strategic-level e-business systems to business strategy, as well as to document patterns of best business practices. They explore goal modeling for providing traceability and alignment between strategic levels (business model and business strategy) and tactical and operational ones (business process model and system requirements). This work is still preliminary, but it provides encouraging insights regarding the scalability of GRL for strategic business issues.

— (Weiss and Esfandiari, 2004) use both GRL and UCMs to model Web services and their interactions. However, their focus is not on modeling business processes per se, but on detecting undesirable interactions among services. We see this work as complementary to ours. It could be used to discover architectural alternatives where they are not apparent, and to strategize about ways of restructuring a business process to meet business goals.

CONCLUSIONS AND FUTURE WORK

In this section, we summarize our contributions, and identify areas of future work.

Contributions

We have made a case for using URN as an approach for business process modeling, and illustrated the approach and its benefits using a supply chain management case study. In the following, our goal is to evaluate the approach by reviewing how URN addresses the requirements for a BPM approach identified in the introduction:

— How does URN address the "W5 questions" (*why*, *what*, *who*, *when*, and *where*)?
— How does it support the analysis of the business model?
— How does it allow multiple stakeholders to participate in the modeling process?

As described, URN comprises two notations: GRL for goal-oriented modeling, and UCM for modeling scenarios. Together, these notations help address the W5 questions:

— *Why* do this activity? GRL models allow the analyst to link business or system goals to architectural alternatives, and thus to document the rationale for a particular activity.
— *What* should this activity be precisely? Starting from high-level business goals, GRL supports their iterative refinement into high-level tasks. Further refinement of these tasks into concrete low-level responsibilities, plug-ins, or scenario definitions can be achieved via a UCM model. In particular, its hierarchical abstraction capability allows us to scale our models to large business processes.
— *Who* is involved in this activity? A UCM model does not only provide a refinement of high-level tasks into low-level responsibilities, but can also capture the structure of the organization supporting the business process. Components can be defined for each role in the business process. Furthermore, component visibility can be restricted as required by nesting them.
— *Where* and *when* should this activity be performed? A UCM model also allows us to define how responsibilities are allocated to components, as well as their temporal ordering via constructs for expressing sequence, choices, concurrency, and synchronization.

Additionally, a business process model should enable ways of (formally) analyzing the processes and goal satisfaction. Analysis of the business process is supported in two ways. The level propagation mechanism in GRL lets us analyze the impact of architectural alternatives (e.g., centralized logging) on high-level goals (of both functional and non-functional nature),

as illustrated in the section on architectural change analysis By capturing architectural reasoning in a GRL model, we can compare alternatives as they apply to a particular context. Unlike use cases, UCM models are not based on an informal textual representation, and can therefore also provide input to various validation activities such as testing and performance analysis. Thus, UCM provides mechanisms for ensuring the validity and traceability of the model, and supports more detailed analysis.

Business process models should be understandable to various stakeholders, including customers. With URN we can model a business process at different levels of formality supporting all development stages from early requirements to early design. A GRL model can describe "soft" aspects of a business process, e.g., high-level business goals such as customer satisfaction, and their refinement into operational goals (e.g., the number of orders handled successfully). The hierarchical abstraction mechanism of UCM models (via stubs and plugins) allows us to hide lower-level details in defining a business process, while preserving a sense of its overall operation. In a UCM model we can simulate scenarios by defining various conditions on the operation of the system, and walking through their execution. The interaction of scenarios and the concurrency properties of a UCM model can be analyzed informally or formally by mapping the UCM model into a suitable target language. In this article, we have illustrated the usefulness of representing UCM scenarios as Message Sequence Charts to support visualization, shared understanding, and analysis. Mappings to other languages exist, as discussed in (Amyot, 2003; Amyot, Weiss, and Logrippo 2004).

Finally, as suggested in the previous section, URN can be integrated partially or entirely into an existing business process modeling approach. It does not have to replace current ways of creating and analyzing models to be useful.

Hence, URN satisfies the requirements of a business modeling language, independently of the application area.

Future Work

In the near-term we expect to work on three research objectives related to the extraction of test goals and test case generation, extraction of service operations, and performance analysis of business processes described in a UCM model:

— Deriving test goals and test cases for the case study. This work will be done in relation to our work on validation described elsewhere (Amyot, Weiss, and Logrippo, 2004).
— Add support to the UCMNAV tool for recognizing messages that are part of the same service operation (e.g., messages that form request/response pairs, or are notification operations in response to an earlier request/response operation).
— Create a performance model for architectural trade-off scenarios.

A longer-term objective is to map UCM models to a Web services implementation. Here we envision work on two research objectives:

— Generate WSDL descriptions from Web services templates.
— Map UCM scenarios to the Business Process Execution Language (BPEL) for Web Services (Andrews, 2003) that can be directly executed by a BPEL execution engine.

Another topic for future research is the possibility of using URN for organizational design and the analysis of business models. One important feature of a UCM model is that the *same* scenarios can be mapped to a different set of components. For example, we could explore the architectural impact of different business models, such as replacing the retail business model with a direct-to-customer business model, by defining a set of components that excludes the retailer, but reusing the scenarios we have already documented. This topic of research is explored in more detail in (Weiss and Amyot, 2005).

Acknowledgements. This work has been supported financially by the Natural Science and Engineering Research Council of Canada, through its Strategic Grants and Discovery Grants programs.

REFERENCES

Amyot, D. (2003). Introduction to the User Requirements Notation: Learning by Example. *Computer Networks*, 42(3), 285-301, 21 June. http://www.usecasemaps.org/pub/ComNet03.pdf

Amyot, D., Cho, D.Y., He, X., & He, Y. (2003). Generating Scenarios from Use Case Map Specifications. *Third International Conference on Quality Software (QSIC'03)*, Dallas, USA, November. http://www.usecasemaps.org/pub/QSIC03.pdf

Amyot, D., Echihabi, A., & He, Y. (2004). UCMEXPORTER: Supporting Scenario Transformations from Use Case Maps. *Proc. of NOTERE'04*, Saïdia, Morocco, June, 390-405.

Amyot, D., Weiss, M., & Logrippo, L. (2004). UCM-Based Generation of Test Goals. *ISSRE'04 Workshop on Integrated-reliability with Telecommunications and UML Languages (WITUL)*, Rennes, France, November 2004.

Andrews, T., Curbera, F., *et al*. (2003). *Business Process Execution Language for Web Services, Version 1.1*, http://www-106.ibm.com/developerworks/library/ws-bpel

Beck, K. (2003). Test-Driven Development: By Example, Addison-Wesley

Bleistein, S.J., Aurum, A., Cox, K., & Ray, P.K. (2003). Linking Requirements Goal Modeling Techniques to Strategic e-Business Patterns and Best Practice. *8th Australian Workshop on Requirements Engineering (AWRE'03)*, Eds. Didar Zowghi & Ban Al-Ani. UTS, Sydney, 13–22. Retrieved August 15, 2004, from http://www.caeser.unsw.edu.au/publications/pdf/Tech03_2.pdf

Buhr, R.J.A. & Casselman, R.S. (1996). *Use Case Maps for Object-Oriented Systems*. Prentice Hall. http://www.usecasemaps.org/pub/UCM_book95.pdf

Buhr, R.J.A. (1998). Use Case Maps as Architectural Entities for Complex Systems. *IEEE Trans. on Software Engineering*, Vol. 24, No. 12, December, 1131-1155. http://www.usecasemaps.org/pub/ucmUpdate.pdf

Carlson, D. (2001). *Modeling XML Applications with UML*. Addison-Wesley, 2001.

Chung, L., Nixon, B.A., Yu, E., & Mylopoulos, J. (2000). *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Dordrecht, USA.

de Bruin, H., and van Vliet, H. (2001). Scenario-based generation and evaluation of software architectures. *Generative and Component-Based Software Engineering (GCSE'01)*, LNCS 2186, Springer.

Endrei, M., Ang. J., Arsanjani, A., *et al.* (2004). *Patterns: Service-Oriented Architecture and Web Services*, IBM Redbook, April 2004. http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf

Gordijn, J. (2002). *Value-based Requirements Engineering: Exploring Innovative e-Commerce Ideas*. Ph.D. thesis, Vrije Universiteit, The Netherlands, SIKS Dissertation Series No. 2002-08. http://www.cs.vu.nl/~gordijn/thesis.htm

Gordijn, J., & Akkermans, J. (2003). Value-based requirements engineering: exploring innovative ecommerce ideas. *Requirements Engineering Journal*, 8:114-135.

ITU-T – International Telecommunications Union (2004). *Recommendation Z.120 (04/04) Message Sequence Chart (MSC)*. Geneva, Switzerland.

ITU-T – International Telecommunications Union (2003). *Recommendation Z.150 (02/03), User Requirements Notation (URN) – Language Requirements and Framework*. Geneva, Switzerland.

Lau, D., & Mylopoulos, J. (2004). Designing Web Services with TROPOS. *IEEE International Conference on Web Services (ICWS'04)*, San Diego, USA, 306-313.

Liu, L., & Yu, E. (2003). Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. *Information Systems (Journal)*, Vol.29, No.2. . http://www.cs.toronto.edu/~liu/publications/

OMG – Object Management Group (2003). *Unified Modeling Language Specification (UML)*, version 1.5, March 2003. http://www.omg.org/uml/

OMG – Object Management Group (2004). *Model Driven Architecture (MDA)*. http://www.omg.org/mda/

Petriu, D.B., & Woodside, C.M. (2002). Software Performance Models from System Scenarios in Use Case Maps. *Computer Performance Evaluation / TOOLS*, LNCS 2324, Springer, 141-158.

Petriu, D.B., Amyot, D., & Woodside, C.M. (2003). Scenario-Based Performance Engineering with UCMNav. *11th SDL Forum (SDL'03)*, Stuttgart, Germany, July. LNCS 2708, 18-35. http://www.usecasemaps.org/pub/SDL03-UCM-LQN.pdf

Telelogic AB (2004). *Tau SDL Suite*. http://www.telelogic.com/products/tau/sdl/index.cfm

UCM User Group (2003). *UCMExporter*. http://ucmexporter.sourceforge.net/

UCM User Group (2004). *UCMNav 2*. http://www.usecasemaps.org/tools/ucmnav/index.shtml

URN Focus Group (2003a). *Draft Rec. Z.151 – Goal-oriented Requirement Language (GRL)*. Geneva, Switzerland, Sept. 2003. http://www.UseCaseMaps.org/urn/. See also http://www.cs.toronto.edu/km/GRL/

URN Focus Group (2003b). *Draft Rec. Z.152 – Use Case Map Notation (UCM)*. Geneva, Switzerland, Sept. 2003. http://www.UseCaseMaps.org/urn/. See also http://www.UseCaseMaps.org/

Weiss, M., & Amyot, D. (2005). Designing and Evolving Business Models with URN. *Montreal Conference on eTechnologies (MCeTech)*, Montréal, Canada, January 2005.

Weiss, M., & Esfandiari, B. (2004). On Feature Interactions among Web Services. *IEEE International Conference on Web Services (ICWS'04)*, San Diego, USA, 88-95.

WS-I – Web Services Interoperability Organization (2003a). *Supply Chain Management: Use Case Model*, Version 1.0. http://www.ws-i.org

WS-I – Web Services Interoperability Organization (2003b). *Supply Chain Management: Sample Application Architecture*, Version 1.0.1. http://www.ws-i.org

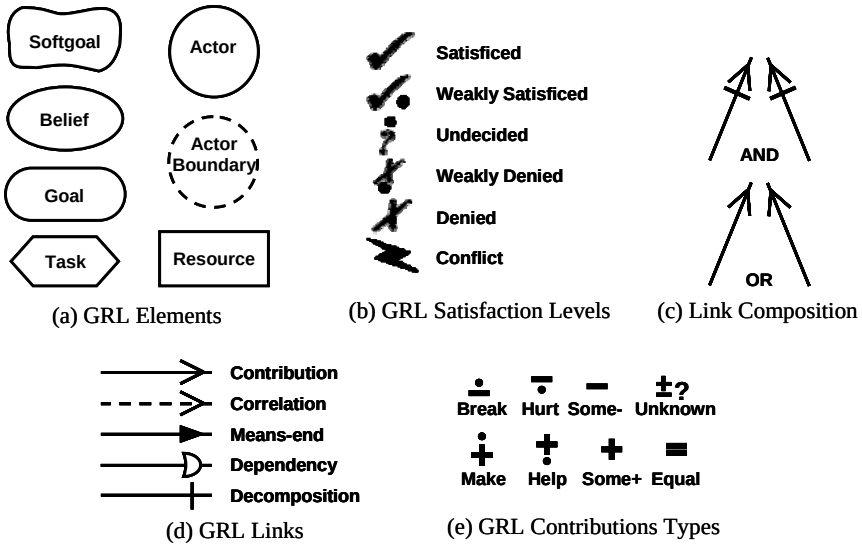
W3C (2001). *Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl

Yu, E. (1997). Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. *3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, Washington, USA, 226-235.

Yu, E., & Liu, L. (2000). *Organization Modelling Environment* (OME). http://www.cs.toronto.edu/km/ome/

Yu, E., & Mylopoulos, J. (1998). Why goal-oriented requirements engineering. *Proceedings of the 4th REFSQ*, Pisa, Italy, 15–22.

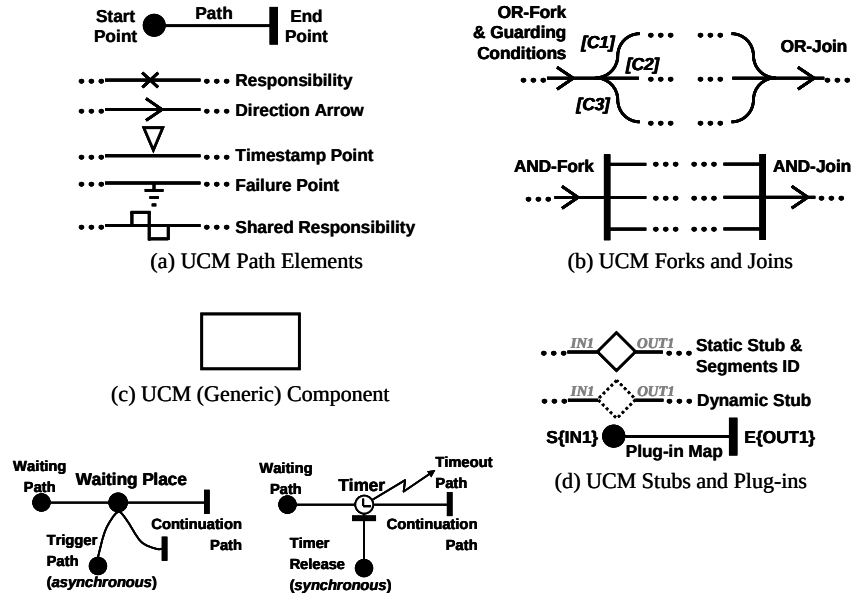
ANNEX A: SUMMARY OF THE USER REQUIREMENTS NOTATION



(a) GRL Elements (b) GRL Satisfaction Levels (c) Link Composition

(d) GRL Links (e) GRL Contributions Types

Fig. 18. Summary of the GRL notation.



(a) UCM Path Elements (b) UCM Forks and Joins

(c) UCM (Generic) Component (d) UCM Stubs and Plug-ins

(e) UCM Waiting Places and Timers

Fig. 19. Summary of (a subset of) the UCM notation.

ANNEX B: PATH CONTROL VARIABLES FOR SCENARIO DEFINITIONS

The following are the global Boolean path control variables used in the supply chain management UCM model :

— *CanAccessLog*: Is the requester allowed to access the event logs?
— *ItemListEmpty*: Is the list of remaining items to order empty?
— *LogRequestValid*: Is the log access request valid?
— *MoreItems*: Are there more items that can be provided by the current warehouse?
— *ProductExists*: Does the requested product exist?
— *ReplenishTimer_timeout*: Will the replenishment timer time out?
— *RepositoryAvailable*: Is the log repository available?
— *SomeItemsShipped*: Are there any items being shipped to the consumer?
— *StockStillSufficient*: Will the stocks be sufficient for the next product (for simulation)?
— *SufficientInventory*: Is the inventory sufficient? (If not, goods need to be manufactured)
— *SufficientStock*: Are the stocks sufficient for the current product?
— *ValidOrder*: Is the order valid?
— *WarehouseLeft*: Any warehouse left to which the remaining items could be ordered?