Designing and Evolving Business Models with URN

M. Weiss*, D. Amyot†
*School of Computer Science, Carleton University
Ottawa, ON, K1S 5B6 (Canada)
Email: weiss@scs.carleton.ca
†SITE, University of Ottawa
800 King Edward
Ottawa, ON, K1N 6N5 (Canada)
Email: damyot@site.uottawa.ca

***Abstract*— The User Requirements Notation (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 emphasis on telecommunications systems and services. However, URN can also be applied to business process modeling. In this paper, we illustrate how goals can help design suitable business processes expressed as scenarios, and the architectures supporting them. URN models can also help us determine when is the right moment to improve the business model. Through a supply chain management case study, this paper illustrates how architectures can evolve in that context while the scenarios remain untouched.**

I. Introduction

In today's rapidly evolving world, companies need to constantly adjust their business models to changes in their environment. However, they also need to do so in a controlled manner. An approach to evolving business models needs to strike a balance between capitalizing on new opportunities, and entering uncharted territories by mitigating the risks involved with such a change. The approach needs to be *lightweight* in order to quickly evaluate alternative models, but must also be reliable.

In this paper, we argue that the *User Requirements Notation* (URN) enables such an approach. While it allows us to explore alternative business models, it addresses the need to preserve investments in existing business processes. Business processes can be expressed as scenarios, which are defined separately from the participants in the business model that perform them. This separation of concerns allows us to experiment with different business models without changing the underlying business processes.

A. Business Process Modeling

Business process modeling (BPM) is a structured method for describing and analyzing opportunities of improving the business objectives of multiple 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 software architecture elements such as Web services) are also desirable. Hence, business models, business processes, and software architectures need to be developed in an integrated manner (Figure 1).

B. User Requirements Notation

The purpose of URN is to support, in a semi-formal and lightweight way, the modeling and analysis of user requirements in the form of goals and scenarios. URN is being standardized by the International Telecommunications Union in the Z.150 series of Recommendations [15]. A brief overview of the notation is presented here, but a more complete introduction can be found in [1][15].

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* [23], summarized in Figure 2. For the last decade, goal-oriented modeling has been a very active field in the requirements engineering community [33]. One of the most well-established language is the NFR (Non-Functional Requirements) framework [9]. GRL includes some of the most interesting concepts found in the NFR framework and complements them with agent modeling concepts from the *i** framework [31]. 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. However, as for most notations, not all requirements are expressible with URN and hence conventional textual requirements can still supplement URN models.

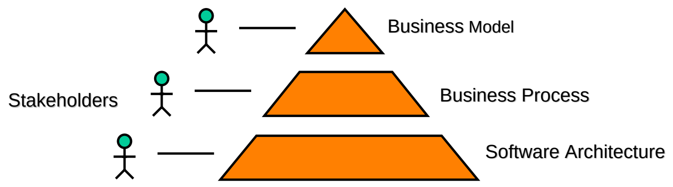
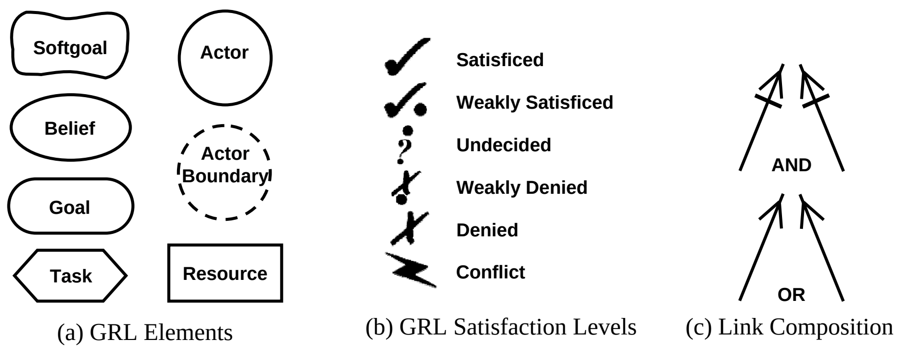


Fig. 1. Three aspects of Business Process Modeling.



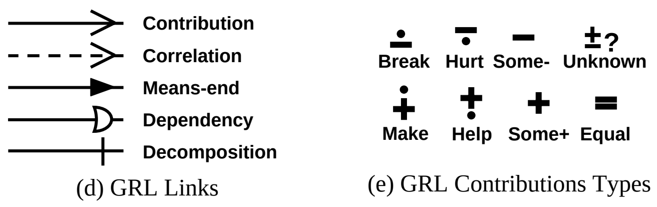


Fig. 2. Summary of the Goal-oriented Requirement Language (GRL).

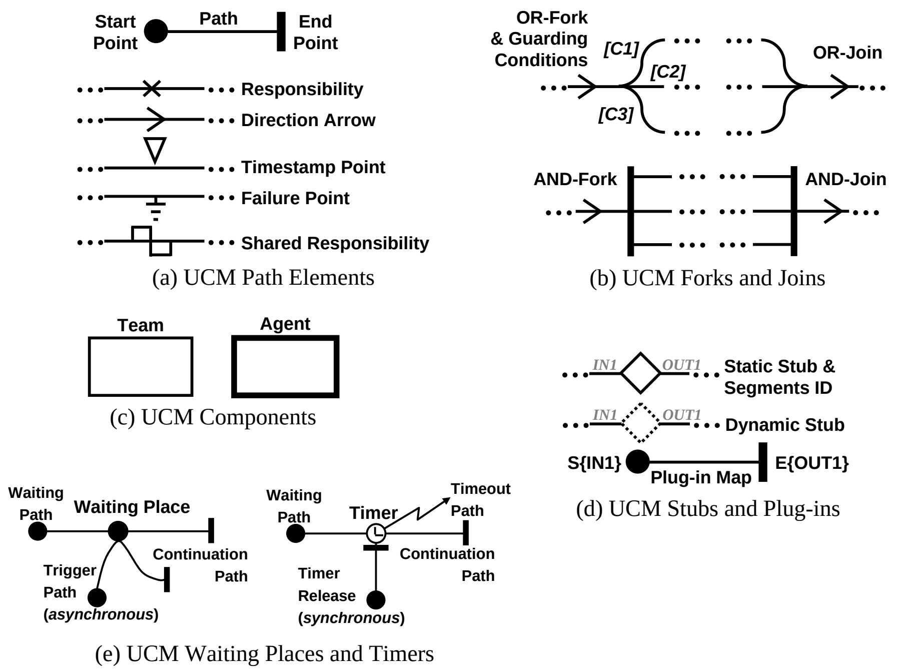


Fig. 3. Summary of the Use Case Map (UCM) notation.

The second part of URN is the *Use Case Map* (UCM) notation, described in [24] and summarized in Figure 3. The UCM notation was first proposed to depict emerging behavioral scenarios during the high-level design of distributed object-oriented reactive systems [7][8]. It was later found to be an appropriate 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).

Note that, in our case study (overviewed in the next section), only some of the symbols found in Figures 2 and 3 are being used, which does not preclude the applicability of the other symbols to business process modeling in general.

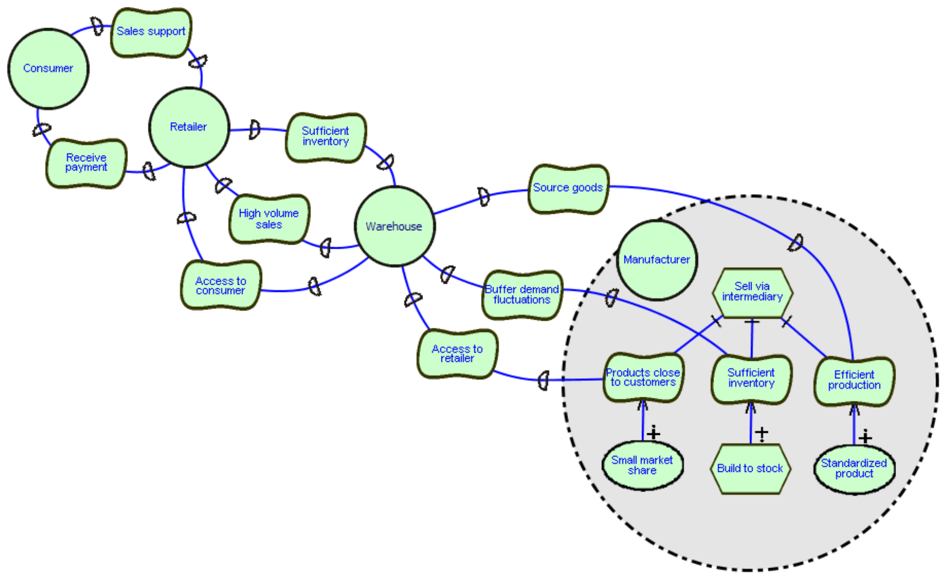


Fig. 4. GRL actor diagram: Sell to stock via warehouse and retailer (R) strategy.

C. Case Study

Our example is based on a WS-I (Web Services Interoperability) case study [28][29]. These documents describe a simple supply chain management system in terms of use cases defining the use of Web services in structured interactions and identifying basic interoperability requirements. A sample deployment architecture is also introduced, although we did not use it to generate our model. Instead, it provided a way of validating that our approach lead to comparable results.

The use cases and architecture include several types of actors: consumers, retailers, warehouses, and manufacturers. The use case model integrates high-level functional requirements, a set of simplifying assumptions, and eight use cases and activity diagrams. Non-functional requirements of the nature considered by URN are not specified in the WS-I case study.

The main high-level functional requirements are:

- Retailer offers electronic goods to Consumers.
- Retailer must manage stock levels in Warehouses.
- Retailer must restock a good from the respective Manufacturer's inventory, if the stock level in one of its Warehouses falls below a certain threshold.
- Manufacturers must execute a production run to build the finished goods, if a good is not in stock.

In a recent contribution [26], we showed how a UCM model could be extracted from such use cases and informal requirements. We argued that URN offers suitable and useful features for modeling and analyzing business processes, and meets the goals of a BPM language.

In this paper, we show how a similar UCM model could be designed given a set of business goals and informal requirements as a starting point (Section II). In Section III, we explore various links between goals and scenarios as well as transformation to detailed scenarios to promote understanding of the system. We then focus on business model design. Various ways to evolve the business model from the Manufacturer's point of view are explored in Section IV. A brief overview of related work and conclusions follow in Sections V and VI.

II. Designing, from Goals to Scenarios

A. Business Goal Model in GRL

Business goals describe the objectives a business should achieve. They provide answers to *why* particular activities are performed in a business process. Business goals can be modeled in GRL. GRL provides a higher, strategic level of modeling both the current situation and future evolution of a business and its environment in terms of goals and their interactions. In this section we will focus on modeling the current business. Business model evolution will be discussed in Section IV.

Figure 4 shows the GRL model for a manufacturer that sells to stock via warehouses (a.k.a. distributors), and retailers. Given the dominant role that the intermediaries (warehouse and, in particular, the retailer) play, we will also refer to this business model as the **R** (Retailer) strategy. This model represents each participant in the business model (consumer, retailer, warehouse, and manufacturer) as an *actor*, and indicates their *dependencies*. Thus, for example, the Consumer depends on the Sales Support provided by the Retailer, whereas the Retailer relies on the Consumer to Receive Payment. The half-moon symbol indicates the direction of the dependency.

Here we adopt the definition of a business model from [25] as a set of participants and the flows between them. The participants include the company whose business model we are describing, its customers, suppliers, and allies or intermediaries. Value is created in the form of information, product, and money flows between the participants. At present, we do not represent the type of flow in our GRL models (the are all expressed in abstract terms as dependencies between actors).

The Manufacturer actor is expanded in the diagram (the actor boundary is shown as a dotted circle partially under to the actor) to reveal its internal *goals*. There are two *tasks* (hexagons) that the manufacturer performs, Sell via intermediary and Build to stock. The Sell via intermediary task is decomposed into three *softgoals* (where softgoals, shown as clouds, are goals that can never be fully satisfied). Tasks, goals, and softgoals can be recursively refined via such decomposition. The manufacturer wants to position its Products close to customers, as supported by the Access to retailer that the warehouses provides. This goal is also guarded with a precondition (Small market share) modeled as a *belief* (ellipse). This allows us to state that the goal is only an appropriate business objective for a manufacturer who does not have a recognizable brand in the marketplace, and a correspondingly large market share.

The manufacturer also ensures Sufficient inventory by building products to be held in inventory (modeled as the task Build to stock). The inventory levels try to anticipate the market demand. However, as there can be unexpected changes in the demand, the manufacturer relies on the warehouse to Buffer demand fluctuations. Furthermore, the manufacturer enjoys Efficient production levels as long as the market demand is for a Standardized product.

Preconditions for this business model are modeled as beliefs and connected to other model elements through *make* contributions. Therefore, the levers for evolving this business model are moves that increase the market share or make the product more differentiated.

B. Scenario Model in UCM

In Figure 4, we have identified several actors which will be shown in UCM models as *agent components* (rectangles with thick lines). They have to be involved in the support of the various functionalities identified in the informal requirements (e.g., Section I-C).

UCM models often start with a single, top-level map called *root map*. One possible root map for our business process is shown in Figure 5. 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.

In the UCM notation, 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. Components are responsible for the various activities (called *responsibilities* and indicated by X's on a path) allocated to them. As a convention here, we use UCM agents (thick lines) to represent GRL *actors*. Additionally, team components (thin lines) are used to capture the various

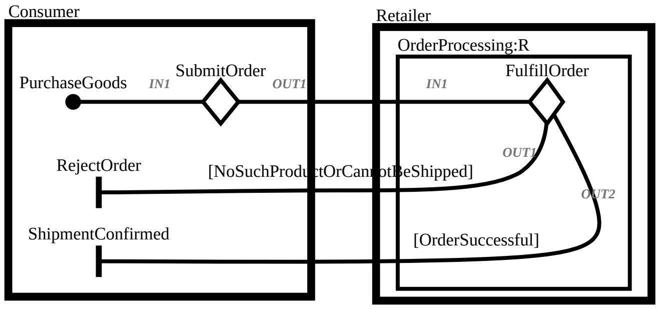


Fig. 5. Sell-to-stock root Use Case Map.

roles an agent can play. Several roles (e.g., OrderProcessing, InventoryManagement, and Production in our example) can be associated with an agent simply by showing component containment. Only the components involved in the current part of the business process need to be shown in a given map. In the following UCM models, we use :R to indicate that a role is associated with a retailer agent, :M for a manufacturer, and :W for a warehouse.

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 Figure 5) 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 Sell-to-stock 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 Figure 6. In FulfillOrder, shown in Figure 7, 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.

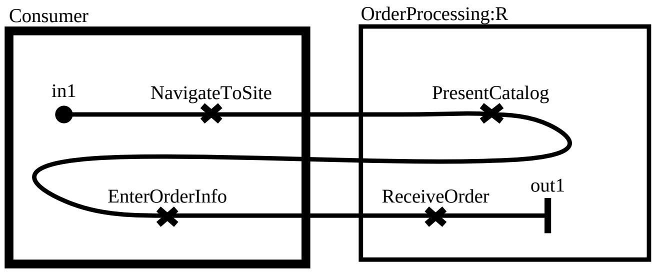


Fig. 6. SumbitOrder plug-in for Fig. 5.

The process of sourcing goods is shown in Figure 8. It is important to note that, in a UCM model, we do not need to map each use case separately, but we can integrate several in the same diagram. The complexity of the resulting model can be reduced through hierarchical abstraction, as provided by stubs and plug-ins.

The retailer tries to locate the ordered goods in its ware-

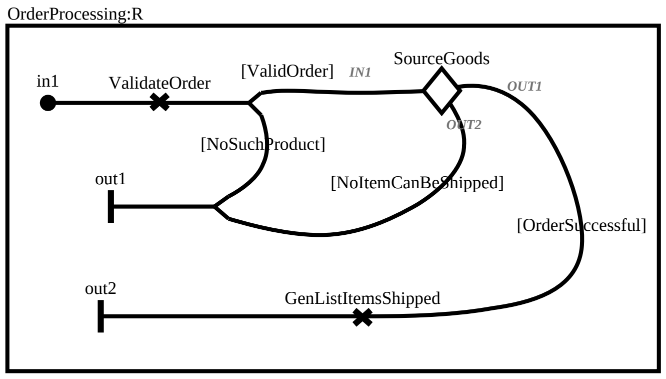


Fig. 7. FulfillOrder plug-in for Fig. 5.

houses. 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. This process results in a list of the items that each warehouse will ship, and accordingly adjusted inventory levels.

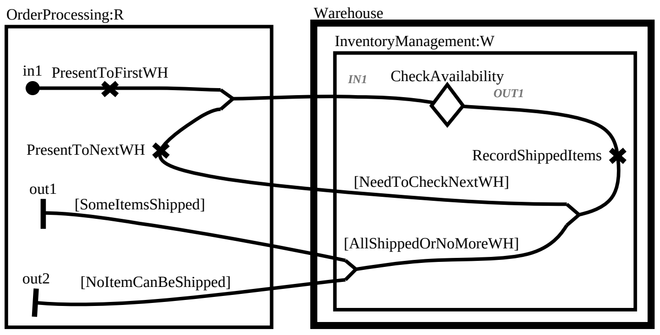


Fig. 8. SourceGoods plug-in for Fig. 7.

The CheckAvailability submap in Figure 9 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. The start point touching the main path is triggered in-passing and this leads to the Replenishment stub. This path is to be executed asynchronously (i.e., in parallel) once the DecrementStock responsibility has been performed. By making Replenishment a *dynamic stub* (shown as a dotted diamond), we can specify a *selection policy* to decide whether the stock needs to be replenished or not. These options would be descriped as two separate plug-ins, with one of them being selected at run-time according to the selection policy.

One plug-in for the Replenishment stub would simply be a straight, pass-trough connection from *IN1* to *OUT1*. The other plug-in is shown in Figure 10. The warehouse orders goods from manufacturers to replenish its own stock for a given product. 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).

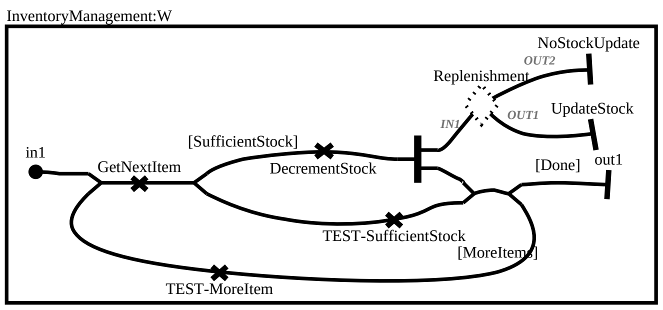


Fig. 9. CheckAvailability plug-in for Fig. 8.

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.

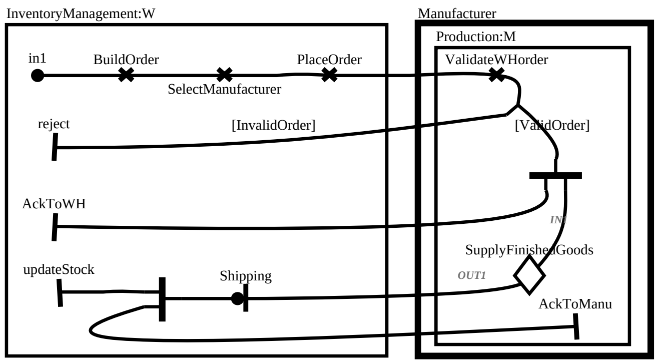


Fig. 10. ReplenishStock plug-in for Fig. 9.

This plug-in also illustrates that bindings to stubs can be fairly flexible (e.g., not all plug-in start/end points and stub input/output segment labels need to be bound). Plug-ins must be bound explicitly to their parent stub(s), although this has not been illustrated so far for simplicity reason. The binding relationship here is described as: {<*IN1* → in1>, <updateStock → *OUT1*>, <reject → *OUT2*>}.

To supply finished goods, we developed the submap described in Figure 11. After receiving 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. This map makes use of a dynamic stub to represent the optional step of manufacturing finished goods (not shown here).

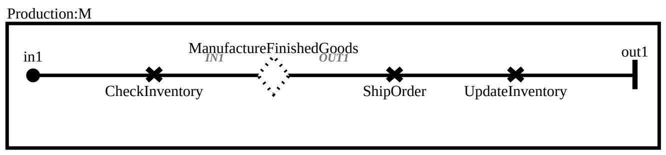


Fig. 11. SupplyFinishedGoods plug-in Fig. 10.

III. Understanding Complex Processes

The understanding of complex business processes is influenced by many aspects. This section discusses two important contributors: the exploitation of links between GRL models and UCM models (the former providing rationales for the latter), and the extraction of specific scenarios from the process.

A. Links Between the GRL and UCM Models

In URN, various traceability links can be created between GRL and UCM models. If a GRL model is fine grained, then detailed elements such as GRL tasks and goals can be linked to specific UCM responsibilities, path segments, scenario definitions or entire plug-in maps. Responsibilities are activities that represent something to be performed (operation, task, function, etc.). GRL goals and softgoals such as those in Figure 4 can be refined into high-level tasks (not shown here), and those tasks into low-level UCM responsibilities. This provides a traceable rationale for the scenarios and their responsabilities, hence explaining *why* they exist and are structured in this way.

From another perspective, UCM models explain *what* the activities related to a business goal are (responsibilities and scenarios), *who* is involved in these activities (actors and components), *where* they are performed (allocation to components), as well as *when* they should be performed (via constructs for expressing sequence, choices, concurrency, timers, and synchronization).

GRL models also allow analysts to link business or system goals to architectural alternatives, and thus to document the rationale for a particular choice. For instance, in [1][26], several ways of allocating UCM responsibilities to components are explored and the decision is based on the contribution of each alternative to the satisfaction of higher-level GRL goals such as performance, reuse of current infrastructures, and maintainability.

Another use for GRL models consists in considering and evaluating different configurations of actors, or allocation of roles to actors. This aspect will be further explored in an evolution context in Section IV.

Although there exists tools to create GRL and UCM models (OME [32] and UCMNav [22] have been used in this paper), these types of links are currently not supported by any tool. However, [20] presents an approach where UCM models are exported to a requirements management system (e.g., Telelogic DOORS) so that traceability links between UCM elements and external requirements and goals can be created, explored, and maintained as the models evolve.

B. Single Scenario in MSC

One problem everyone faces when studying a complex UCM model is that we need to flip back and forth between many maps, nested through stub/plug-in relationships. This hinders the understandability of specific scenarios. However, this issue can be overcome by extracting specific scenarios and representing them using a suitable notation.

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 in guards attached to OR-forks and in selection policies (in dynamic stubs). 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 were created and used to formalize the various conditions found in the model.

A UCM model may also include several groups of *scenario definitions*. Each such definition consists of initial values for the variables, a set of start points initially triggered, and an optional post-condition expected to be satisfied at the end of the execution of the scenario. In our model, we created 20 scenarios definitions categorized in five groups. These scenarios cover the interesting functionalities offered by the system, as well as all the UCM path segments in the model.

Scenario definitions can be combined to a path traversal algorithm in order to highlight specific scenarios in a complex UCM model, or to transform them to other representations. Details of the various algorithm used here can be found in [2][24]. In a nutshell, the 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.

The target representation selected here is Message Sequence Charts (MSCs) [14]. The UCM model was first constructed using the UCMNav tool [22]. Then, scenario definitions were added and this tool generated the resulting scenarios in a XML format whose schema is described in [3]. Another tool (UCMExporter [3]) takes these scenarios in XML and converts them to MSCs or to UML sequence diagrams [18].

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. Actions (small boxes), conditions (hexagons), and concurrent behavior (large boxes crossing many instances) are also supported.

Figure 12 provides a simple example of an MSC generated from a basic scenario where the warehouse has the desired item and the shipment is confirmed, but during the replenishment the inventory is found to be insufficient and hence manufacturing gets involved. This scenario was selected because it traverses all the UCM found in Figures 5 to 11. To preserve the semantics of UCMs and traceability to the original model, UCM components are mapped to MSC instances, start and end points to messages, condition labels to conditions, and responsibilities to actions.

In this MSC, we can observe that a Navigate message shows up while it is absent from the UCM model. This message is synthesized automatically by UCMEXPORTER in order to preserve the causal flow between successive responsibilities found in two different components (Customer and Retailer). 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 message names used in corresponding Web service operations. Plug-ins selected in dynamic stubs are reported as conditions in the MSC to indicate which one was chosen (e.g., the Default plug-in was selected in the dynamic stub that follows CheckInventory from Figure 11). Start points and end points that are not used as connectors in a stub/plug-in binding correspond to messages (e.g., AckToManu and AckToWH from Figure 10).

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 eight 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).

It should also be noted that the MSCs generated here are comparable in content to the sequence diagrams found in [29], 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 MSC or sequence diagrams are created manually.

Scenario definitions, path traversals, and transformations to other formalisms are not limited to model understanding. They also contribute greatly to many validation activities:

- The UCM model itself can be *simulated*. Scenario definitions can be seen as test cases that can be used to ensure nothing is broken as the business process and architecture evolve, in a way somewhat compatible with the test-development approach proposed by the *agile development* community. 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 reach agreement on issues identified in GRL models.
- Test goals can be generated from these scenarios. Several approaches are surveyed in [4].
- Performance annotations can be added to UCM models in order to generate analyzable performance models [20].

The generation of detailed scenarios from higher-level UCM descriptions respects the spirit of model-driven development. In OMG's *Model Driven Architecture* [19], platform-independent models are refined into models containing platform-specific information. Indeed, as seen in this section,

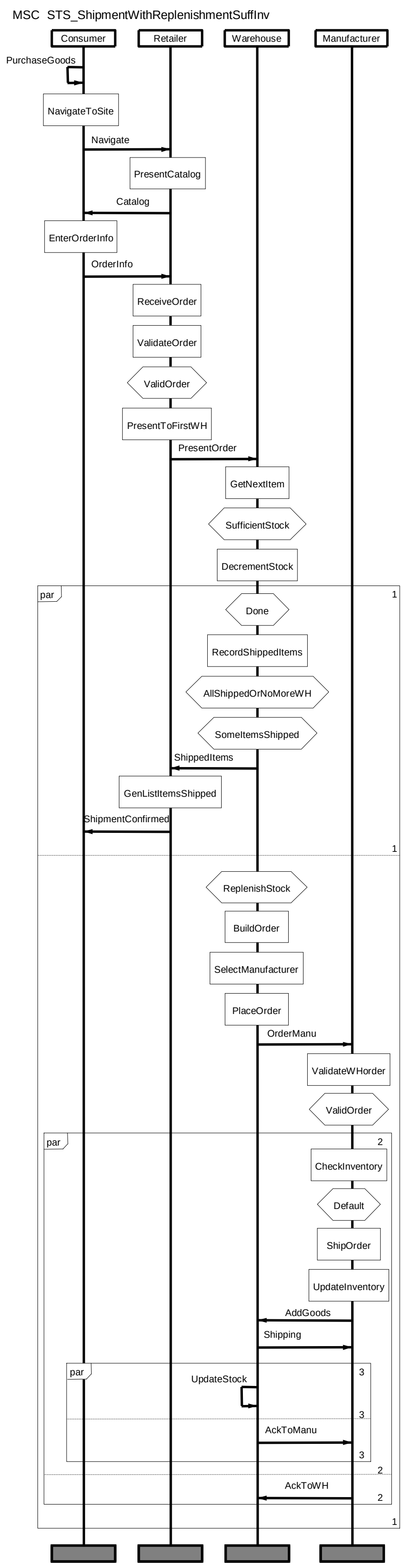


Fig. 12. An MSC scenario for Sell-to-stock strategy.

platform-specific communication information can be added to the scenarios generated from a UCM model.

IV. Evolving Business Processes

A. Evolution of Business Goals

Our working hypothesis in this paper is that we can use the *same* scenario to describe *different* business models and to reason about them. The fundamental underlying concept of UCM is the separation of the definition of a scenario from its allocation to components. Allocations can be reasoned about, and compared using GRL models. This concept lays the basis for an incremental *evolution* of the business model.

Consider the options available to a manufacturer who currently sells its products via intermediaries. As indicated in Figure 4, the manufacturer could consider actions that result in either one or both of the preconditions Small market share and Standardize product to change. Exploring those options leads to several possible evolutions of the business model illustrated by Figure 13.

The boxes in Figure 13 correspond to different business model available to the manufacturer. The initial option (**R**) represents the manufacturer's current model. We name it for the dominant role played by the retailer in controlling access to the customer in this model. This option is exemplified by North-American PC retailers such as CompuSmart and Micro Warehouse.

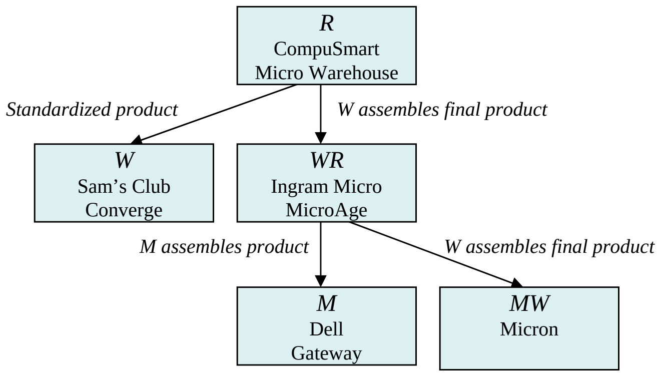


Fig. 13. Different ways for a manufacturer to sell its products.

The arrows indicate the evolution between these business models, and the labels on the arrows characterize the nature of the transition between the models. For example, the transitions from **R** to **W**, and **R** to **WR**, are both about increasing market share. However, in the transition from **R** to **W** the manufacturer keeps selling a standardized product, whereas in the other transition, it can offer a differentiated product. It is the warehouse that assembles the customized product. In both options the warehouse keeps control of order processing.

The manufacturer could increase its market share by partnering with a warehouse. This leads to either the **W** (Warehouse), the **WR** (Warehouse-Retailer), or the **MW** (Manufacturer-Warehouse) strategy. In the first option (**W**), the warehouse now owns the relationship with the customer, and its impact on the manufacturer is in many ways similar to that of the **R** strategy. However, a higher revenue can be expected due to the reduced length of the supply chain. In this option, the manufacturer keeps selling a standardized product. Examples of warehouses that sell direct to customers are Sam's Club (Walmart's warehouse outlets), or the Converge consortium.

In the second option (**WR**), the warehouse assumes additional responsibilities such as assembly of all or part of the product. The main difference from the **W** strategy is that manufacturer can now (via the distributor or warehouse) offer a customized product, and can strengthen its market position against competitors that continue to sell standardized goods. In common with the first option, however, order processing is still performed by the warehouse, which therefore controls the flow of customer information to the manufacturer. This strategy is illustrated by examples such as Ingram Micro or MicroAge, both established distributors of PC components, and providers of value-added services.

Of greater interest to the manufacturer, however, should be the third option (**MW**). In this strategy, illustrated by companies such as Micron, the manufacturer is in the driver's seat. It sells its products directly to the customer, but, in part to share revenue risks, and in part to leverage the distribution experience of a warehouse partner, it outsources distribution to a warehouse. Traditional shipping service providers such as Micron's partner FedEx have developed additional capabilities to manage the inventories of their clients. Although not specialists in the manufacturer's domain, they may still perform some of the assembly tasks formerly performed by the manufacturer, or domain experts such as Ingram Micro. FedEx's service is branded as Merge-in-Transit, and builds on FedEx's extensive distribution network.

The most evolved of these strategies, however, is to assume all key responsibilities (order processing, inventory management, and production) within the manufacturer. This is labeled as the **M** (Manufacturer) strategy in Figure 13. Note that this option does not necessarily imply that the manufacturer handles the physical product, but refers to the control the manufacturer exerts over the information flow in the supply chain. The fully virtual version of this business model (not discussed here) is also known as Value-Net-Integrator [25].

The impact of choosing one of these alternatives can be analyzed within an actor diagram. The GRL model for the **M** strategy is shown in Figure 14. This figure shows that the benefit of selling direct with an internal warehouse allows the manufacturer to Provide tailored services, know the customer and achieve high rates of Repeat business (Own customer relationship), and sell at a Low price while still realizing a high margin. The latter can be achieved by only assembling a product on receipt of a firm order (Build to order), and efficiencies in inventory levels (Low inventory), as well as the float resulting from receiving Advance payment.

However, adopting the **M** strategy is predicated on two preconditions: that the manufacturer already has a Large market share, and the capability to offer a Differentiated product. One important component of our approach is, therefore, to use

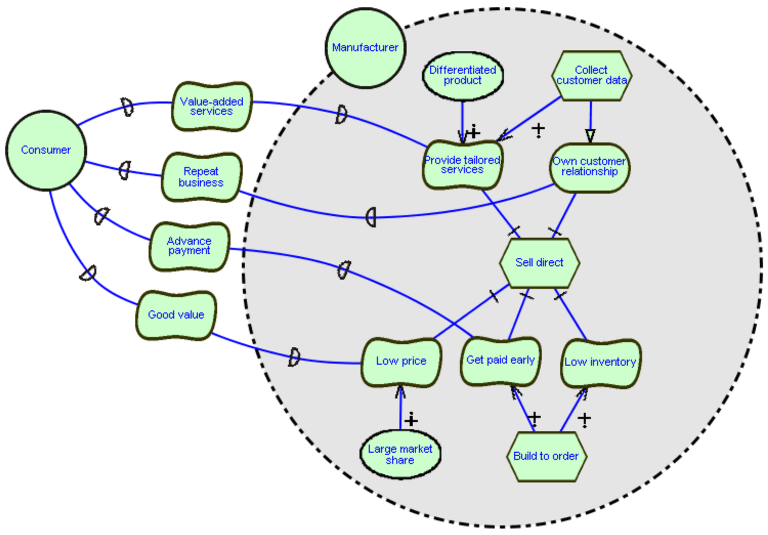


Fig. 14. GRL actor diagram: Sell direct to consumer with internal warehouse (M) strategy.

GRL beliefs to model preconditions for achieving business objectives represented as softgoals. (Similarly, we will later use decomposition of GRL tasks into softgoals to express preconditions for alternative business models to be compared by using GRL models.)

Figure 15 summarizes the business architectures corresponding to these business model alternatives. Since the **W** and **WR** strategies share most aspects, except for the fact that **W** involves standardized, and **WR** differentiated products, their architectures are the essentially the same.

Another GRL model can be used to compare the business models. For the sake of understandability, we limit ourselves to a comparison of the two extreme models, i.e., the **R** and **M** strategies. Figure 16 shows the impact of choosing either alternative on profitability and risk.

The comparison thus focuses on two high-level goals that can be used to characterize any business (not just the business models discussed) of achieving High Profitability and Low Risk. The profitability goal can be achieved by increasing revenue (High revenue), or reducing cost (Low cost). The contributing factors of these goals are the five subgoals of Sell direct identified in Figure 14. The diagram also captures the preconditions for each option via softgoals into which the **M** and **R** tasks are decomposed. These are the same as the beliefs identified earlier.

However, not all companies will be able to evolve their business models as rapidly as they would like to. Figure 16 indicates a key obstacle for evolving quickly from the **R** strategy to the **M** strategy for manufacturers with existing resellers. Trying to remove those resellers from the chain will (initially, at least) lead to a *channel conflict*, and to a loss in sales as the manufacturer can no longer count on the sales from its resellers. Thus, the strength of the existing resale channels is a key determinant for how fast the manufacturer can evolve its business model. In the model, this is represented as a belief (Existing channels) that provides a precondition to Channel conflict.

B. *Evolving the UCM Model*

Evolving a UCM model usually involves modifications to the path elements (including responsibilities), the component architecture, and the allocation of path elements to components. However, in order to evolve our business process from a Sell to stock via warehouse and retailer (**R**) strategy to a Sell direct to consumer with internal warehouse (**M**) strategy, there is little need for modifying the existing UCM paths. Obviously, the underlying component model changes from Figure 15a) to Figure 15c). Thus, two actors are removed, and the three roles (OrderProcessing, InventoryManagement, and Production) become allocated to the remaining Manufacturer component. Accordingly, the new root map shows the Consumer submitting her order to the Manufacturer (see Figure 17).

Note that, for this example, we do not need to modify the allocation of responsibilities to roles, nor do we need to modify the paths themselves. Only the deployment of roles to actors needs to be updated. This is a major benefit of the UCM notation: the scenarios are often robust and long-lived, even when the underlying architecture changes. This is usually not the case with message-based scenario notations like MSCs and UML sequence diagrams.

In the set of maps previously defined for the (**R**) strategy, only two other maps need to be slightly modified to describe the (**M**) strategy. Figure 8 becomes Figure 18 whereas Figure 10 is changed to Figure 19. These modifications are

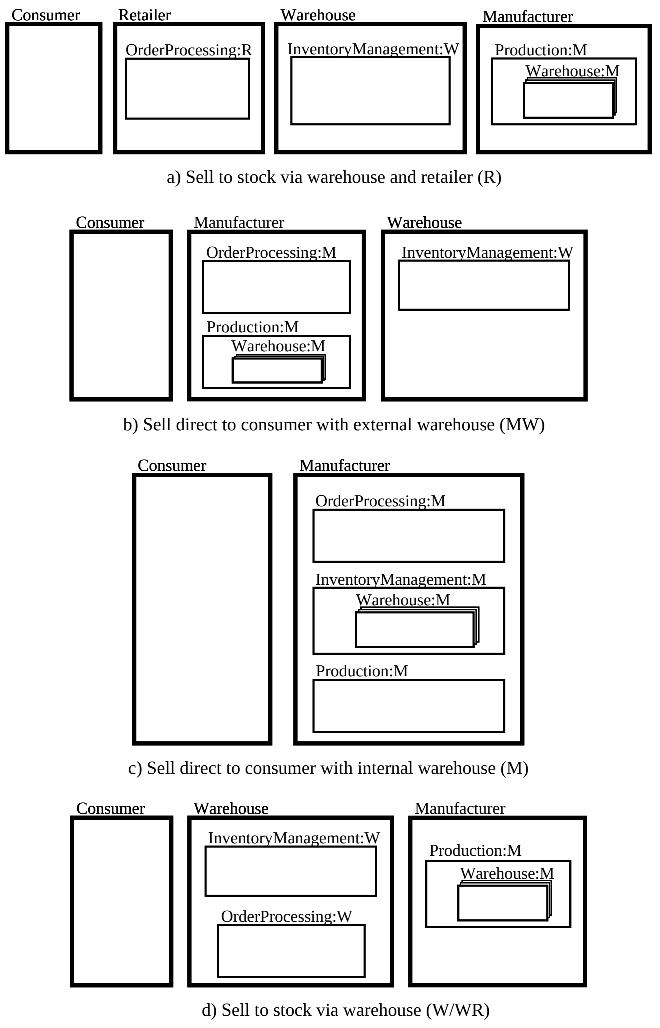


Fig. 15. Four alternative architectures for supporting the business process.

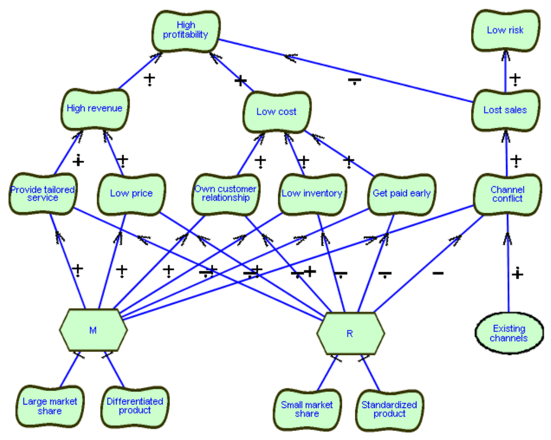


Fig. 16. GRL diagram: Comparison between strategies M and R.

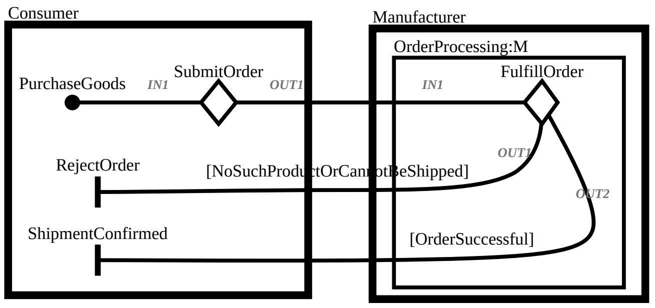


Fig. 17. Strategy M: root Use Case Map.

required because these UCMs showed situations where the causality involved two different actors. In the UCM notation, components in plug-ins do not need to be repeated because, unless stated otherwise, the roles in a submap are allocated to the actor that contains the stub where this plug-in is used. For instance, the OrderProcessing role in Figure 7 is part of the Retailer component because the latter contains the FulfillOrder stub where this plug-in is used. However, in the new (**M**) strategy, this same role is allocated to the Manufacturer component, which now contains the FulfillOrder stub (Figure 17).

Note that in Figure 19, we are keeping the SelectManufacturer and PlaceOrder responsibilities for simplicity, although their interpretation is slightly different (i.e., there is only one manufacturer to choose from). In general, some adjustments to a few such responsibilities, which are specific to a given architecture, may be required in UCM models.

C. *MSC Scenarios Revisited*

The scenario definitions developed for the first UCM model can be reused as is in the second one because the start points and the Boolean variables used in guards and selection policies have not changed. The new model can be simulated again and the resulting scenarios transformed to MSCs.

Figure 20(a) shows the result of exploring the same scenario definition as the one used in Figure 12. It is very similar to the first one because this version uses the roles as MSC instances, and they match the actors used in Figure 12. However, the same scenario could be represented at the actor level only (i.e., without references to the roles). For instance, Figure 20(b) clearly illustrates the various activities for which the Manufacturer is responsible.

Since there are fewer intermediaries, in total there are likely fewer services that will be supported between actors. The interfaces between the Retailer and the Warehouse and between the Warehouse and the Manufacturer are no longer required in this new business process. In fact, the messages on these interfaces have disappeared from Figure 20(b). This could lead, for instance, to simplified Web services required to implement the overall process.

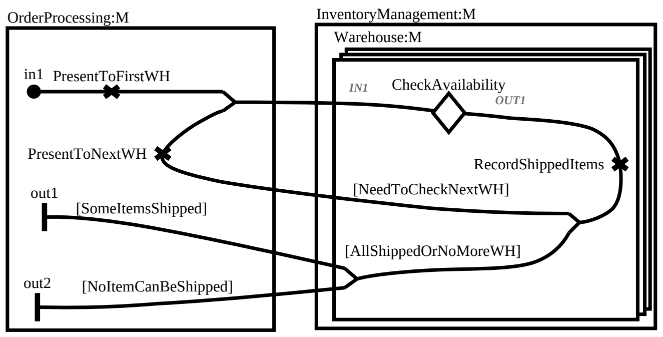


Fig. 18. Strategy M: SourceGoods plug-in.

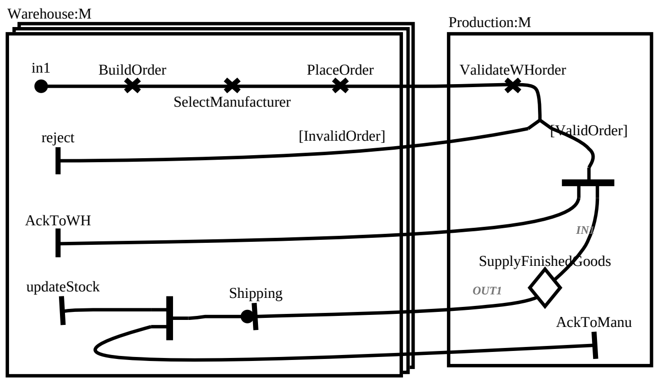


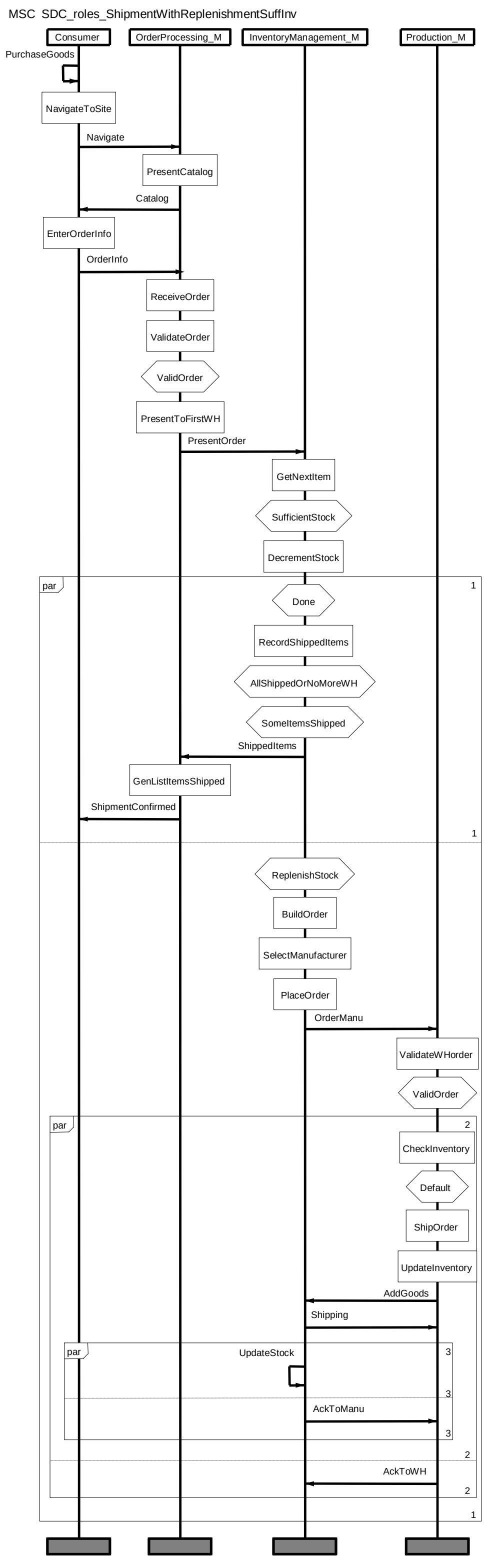
Fig. 19. Strategy M: ReplenishStock plug-in.

V. Related Approaches

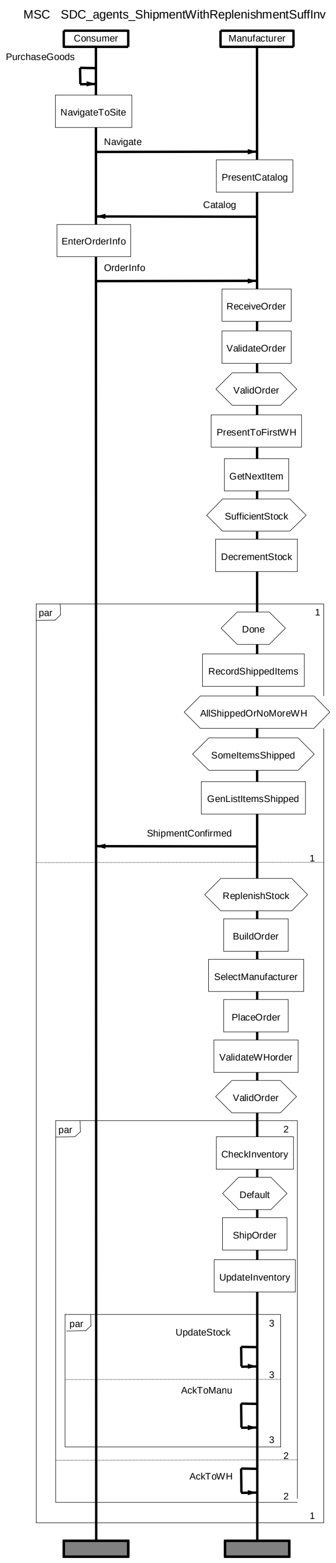
The User Requirements Notation and similar languages have been exploited in various contexts, some of which are related to the one presented here.

A. Other Similar Notations

- In 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 known drawbacks that can be averted by using UCMs to model the early requirements of a business process. Using *extends* and *includes* relationships is 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 seldom provide notions of modeling design goals and linking them to other design artifacts, as in URN.
- The Service-Oriented Architecture (SOA) approach proposed by [11] aims to align services with business goals. In this approach, services are large-grained activities at the use-case level. SOA aims to support many design activities, including domain decomposition, goal-service model creation, and subsystem analysis. 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 or *e3-value* [12][13] 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. *e3-value* uses UCM scenarios to model and analyze revenue streams as causal flows, and to integrate various viewpoints: value, process, and information system. *e3-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. On one hand, *e3-value* is hence much more specific in scope than URN. On the other hand, we can think of *e3-value* as an intermediary view between general GRL models and operational UCM models. Both approaches could therefore be integrated for modeling e-commerce systems.
- Many GRL concepts are formalized in the TROPOS



(a) With roles as instances



(b) With agents as instances

Fig. 20. MSCs for a Sell direct to consumer with internal warehouse (M) strategy.

agent-oriented methodology. In [16], Mylopoulos and Lau explore the use of TROPOS in the context of Web service design, with an emphasis on actors and their dependencies (as was presented here). It is suggested to use the Agent-based Unified Modeling Language (AUML) to refine the goal and actor models prior to defining Web services in WSDL. However, this step is not well illustrated.

B. Other URN Usages

- UCMs are used in [10] 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 in [17] 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. They illustrate their approach with a Web-based training system.
- In [6], Bleistein *et al.* use GRL to link requirements for strategic-level e-business systems to business strategy, as well as documenting recurring 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.
- Both GRL and UCMs are used in [27] 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 our work. 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 user (or company) goals.

VI. Conclusion

This paper introduced how the User Requirements Notation can be used to design and evolve business models. Through a case study, we illustrated the systematic and incremental evolution of (a family of) business model alternatives. GRL models allow the analyst to model the business goals, the specific benefits and liabilities (risks) of each alternative, as well as the dependencies between all participants in the supply chain. We decided to model preconditions as GRL beliefs, which help assess the applicability of business models, and select among different alternative business models.

While GRL models provide rationales, UCM models focus more on the operational aspects of the business process by describing, in abstract terms, who should do what, when, and where. UCMs can integrate multiple scenarios and use cases in a collection of interrelated maps. The notation promotes the evolution of the model by allowing analysts to map responsibilities to components as well as roles to agents/actors in various ways, while minimizing the impact on the rest of the model.

In our case study, we used a standard example (sell through resellers), and developed several alternatives based on the exact same scenario. We discussed the impact of the business models from the perspective of the Manufacturer, who would like to determine when would be a good time to eliminate intermediaries such as Retailers and Warehouses, as well as how this would affect current ways of delivering services.

Although various links between GRL and UCM models can improve the understanding of these two views, sometimes business processes modeled as UCMs become rather complex. They can then be more suitably shown with a different notation, such as MSCs. UCMs ease the design and evolution of an integrated set of scenarios as a whole, but transformations of specific scenarios to linear representations such as MSCs help understand a scenario from end to end without having to go back and forth between different UCMs. Extracting MSCs (or the like) from UCMs also pave the way towards more detailed design and testing activities, in accordance with the requirements of the business process.

We foresee several items for future work:

- Definition of more detailed business processes to enable the mapping of UCM models to Web service implementations. In particular, we would first consider a mapping from UCMs to the Business Process Execution Language (BPEL) for Web Services [5] that can be directly executed by a BPEL execution engine.
- Derivation of test goals from UCM models and transformation to a suitable format for testing, e.g., Web services.
- Look at ways of describing a complex business process as a composition of smaller business processes, to make them more usable as the complexity of the model increases.

Acknowledgment

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

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

[2] D. Amyot, D.Y. Cho, X. He, and Y. He, "Generating Scenarios from Use Case Map Specifications", *Third Int. Conf. on Quality Software (QSIC'03),* Dallas, USA, November 2003. http://www.usecasemaps.org/pub/QSIC03.pdf

[3] D. Amyot, A. Echihabi, and Y. He, "UCMExporter: Supporting Scenario Transformations from Use Case Maps", *Proc. of NOTERE'04*, Saïdia, Morocco, June 2004.

[4] D. Amyot, M. Weiss, and L. Logrippo, “UCM-Based Generation of Test Goals”, *ISSRE04 Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL)*, Rennes, France, November 2004.

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

[6] S.J. Bleistein, A. Aurum, K. Cox, and P.K. Ray, “Linking Requirements Goal Modeling Techniques to Strategic e-Business Patterns and Best Practice”, *8th Australian Workshop on Requirements Engineering (AWRE’03)*, UTS, Sydney, 2003, 13–22. http://www.caeser.unsw.edu.au/publications/pdf/Tech03_2.pdf

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

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

[9] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Dordrecht, USA, 2000.

[10] H. de Bruin and H. van Vliet, “Scenario-based generation and evaluation of software architectures”. *Generative and Component-Based Software Engineering (GCSE’01)*, 2001, LNCS 2186.

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

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

[13] J. Gordijn, J. Akkermans, “Value-based requirements engineering: exploring innovative ecommerce ideas”. *Requirements Engineering Journal*, 2003, 8:114-135.

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

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

[16] D. Lau and J. Mylopoulos, “Designing Web Services with TROPOS”. *IEEE International Conference on Web Services (ICWS’04)*, San Diego, USA, 306–313, 2004.

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

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

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

[20] D.B. Petriu, D. Amyot, M. Woodside, and B. Jiang, “Traceability and Evaluation in Scenario Analysis by Use Case Maps”. To appear in: *Scenarios: Models, Algorithms and Tools*, LNCS, Springer, 2004.

[21] UCM User Group, *UCMExporter*, 2003. http://ucmexporter.sourceforge.net/

[22] UCM User Group, *UCMNav 2*, 2004. http://www.usecasemaps.org/tools/ucmnav/index.shtml

[23] URN Focus Group, *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/

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

[25] P. Weill and M. Vitale, *Place to Space*, Harvard Business School Press, 2001.

[26] M. Weiss and D. Amyot, “Business Process Modeling with URN”. Submitted.

[27] M. Weiss and B. Esfandiari, “On Feature Interactions among Web Services”. *IEEE Int. Conf. on Web Services (ICWS’04)*, San Diego, USA, 2004, 88–95.

[28] WS-I – Web Services Interoperability Organization, *Supply Chain Management: Use Case Model, Version 1.0*, 2003. http://www.ws-i.org/Documents.aspx

[29] WS-I – Web Services Interoperability Organization, *Supply Chain Management: Sample Application Architecture, Version 1.0.1*, 2003. http://www.ws-i.org/Documents.aspx

[30] W3C, *Web Services Description Language (WSDL) 1.1*, http://www.w3.org/TR/wsdl

[31] E. Yu, “Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering”. *3rd IEEE Int. Symp. on Requirements Engineering (RE’97)*, 1997, Washington, USA, 226–235.

[32] E. Yu and L. Liu, *Organization Modelling Environment (OME)*. http://www.cs.toronto.edu/km/ome/

[33] E. Yu and J. Mylopoulos, “Why goal-oriented requirements engineering”. *Proc. of the 4th REFSQ*, Pisa, Italy, 1998, 15–22.