Towards Agile Service-oriented Business Systems: A Directive-oriented Pattern Analysis Approach

Soo Ling Lim[1] Fuyuki Ishikawa[2] Eric Platon[2] Karl Cox[3]

[1] University of New South Wales and NICTA, Sydney, Australia
slim@cse.unsw.edu.au
[2] National Institute of Informatics, Tokyo, Japan
{f-ishikawa,platon}@nii.ac.jp
[3] NICTA, Sydney, Australia
Karl.Cox@nicta.com.au

Technical Report
UNSW-CSE-TR-0806
February 2008

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Volatile requirements should be managed such that changes can be introduced into the system in a quick and structured way. This paper presents Directive-oriented Pattern Analysis (DoPA), a requirements engineering approach that handles volatile requirements by managing the coupling between business intentions and service integration. The key insight is to utilise services as commodities via service choreography patterns. DoPA captures differentiating enterprise intentions as Directives, while using patterns to handle common business needs. This enables the notion of declarative configuration of services to achieve business agility.

1 Introduction

Business software systems face the challenge of rapidly changing requirements. Enterprises want to build, configure and deploy their applications "in real-time" to meet the needs of volatile market conditions [25]. Hence, it is important for the requirements management of these applications to support quick and structured introduction as well as modification of requirements.

Managing requirements change has always been a challenge in requirements engineering (RE) research [15][5]. Existing research isolates requirements that are most likely to change. The techniques used include aspect-oriented [13] and scenario based [3]. However, these methods are business specific; hence unsuitable for services, which are cross-domains and inter-enterprise.

In service-oriented computing (SOC), enterprises use services as fundamental elements for developing their business systems [17]. SOC builds on service-oriented architecture (SOA) where business agility is enabled through business process integration and reuse [10]. Core to SOA is service composition, where a business application combines services into business processes [18]. Also, the application's behaviour can be configured and customised for a specific business context [10]. With the open, rapid and low-cost service compositions offered by SOC, how can requirements for service-oriented systems be managed to respond quickly to business change?

To build, modify and deploy service-oriented systems "in real-time", businesses should utilise services as commodities. Many business services have pervasive and well-defined functionalities [4], such as flight booking, currency exchange, and credit card charging. Applications that are built from these services can be modelled such that volatile requirements are externalised as easily reconfigurable metadata, followed by compositions. Requirements changes can then be cascaded into either a configuration or composition change.

This paper presents DoPA (Directive-oriented Pattern Analysis): a RE approach that manages requirements change by associating enduring requirements with reusable patterns and externalizing volatile requirements as Directives. DoPA associates the functional logic of a system with business archetype patterns [1] which act as placeholders for services. These services can be replaced and reconfigured when the requirements changes. Directives are volatile elements in a business plan [22] that reflect business decisions for addressing change. They capture business flow logic and can be categorised as Business Policies and Business Rules. Structured change occurs when Business Policies are introduced into the system via configuration, while Business Rules via composition. Requirements are managed from the perspective of change and adaptability, resulting in agile systems that can be modified and deployed quickly.

The remainder of the paper is organised as follows. Section 2 discusses background work. Section 3 introduces DoPA. Section 4 validates the DoPA approach with an example case. Section 5 evaluates the approach. Section 6 discusses related work. Section 7 concludes the paper and provides directions for future work.

2 Background

2.1 Services and their commoditisation

SOA encourages reuse by the encapsulation of service functions and the loose coupling of service interactions. In SOC, abstraction emerges in three layers: configuration, composition and customisation. Configuration involves creating service specifications based on service choreography patterns, and cascading requirements change into existing specifications. Composition enables services to be composed from other services via orchestration or choreography [19]. Customisation involves creating new services or modifying existing services.

To achieve business agility, services should be modelled to be configurable for meeting volatile requirements. As configurations are dynamic [10], changing the system in response to requirements can be quick and easy. Composition should model how business services are delivered, as the adoption of business processes is often evolutionary in nature and do not change frequently. Customisation should only be done on services that the business delivers and not on those it uses.

In business service choreography, the development of business process flow logic is separated from functional logic [14]. Business service choreography patterns govern services to interact across business processes, enabling them to be aggregated into workflows. Such patterns promote utilisation of services as commodities by abstracting common functional flow logic into best practices and solutions to meet functional requirements.

2.2 Business archetype patterns

Business archetype patterns [1] are patterns for enterprise computing that address requirements analysis and ontological issues in the business environment. Most businesses are made up of the same semantic elements such as customer, product, and order. These pervasive elements span business domains. Hence, most business systems can be assembled from a sufficiently complete set of archetype patterns. The following paragraphs explain the main concepts of business archetype patterns: archetype; archetype pattern; archetype and archetype pattern variability; and pleomorph.

Archetype is a primordial element that occurs consistently and universally in business environments and business software systems. Archetypes are represented as UML classes and relate to one another via inheritance, association, aggregation and composition. In Figure 2.1, the `ProductType` archetype has an aggregation relationship with the `ProductInstance` archetype.

Archetype pattern is the collaboration between archetypes that occurs consistently and universally in business environments and software systems. The archetypes in Figure 2.1 are part of the `Product` archetype pattern. Other archetypes in the `Product` archetype pattern include `Price` and `ProductCatalogue`. An archetype pattern can be represented as a parameterised collaboration in UML. Figure 2.2 specifies placeholders such as `ProductCatalog` and `Price` by naming them as parameters to the pattern. The instantiation of the pattern involves providing classifiers to play the roles of the placeholders. The classifiers must accord with the semantic constraints of the pattern and its parameters.

Variation can occur in archetypes and archetype patterns to be effective

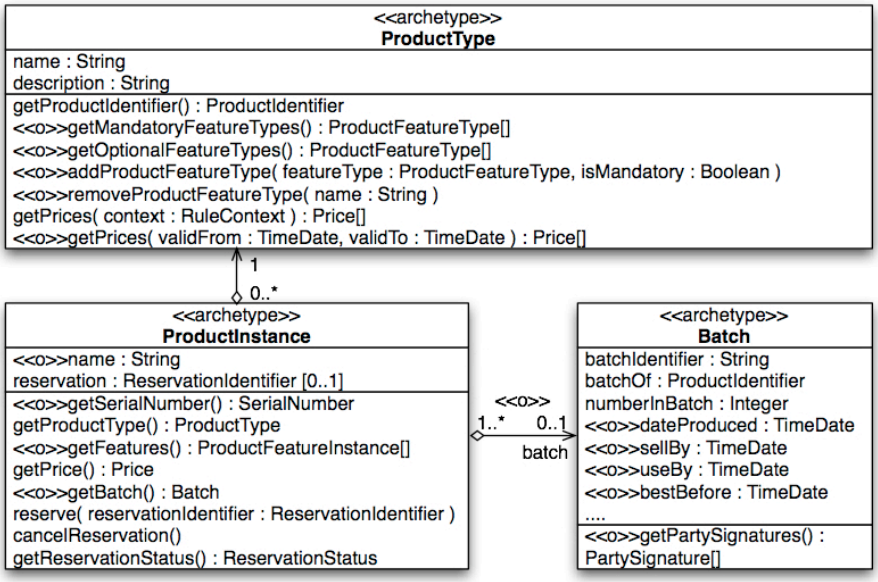

Figure 2.1: Archetypes in the `Product` archetype pattern [1]

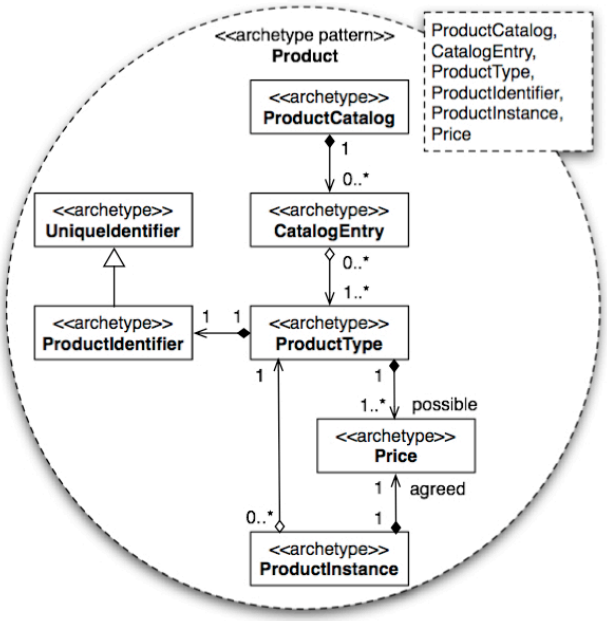

Figure 2.2: Parameterized collaboration [1]

in different business contexts. Archetypes vary in their features (attributes, operations, constraints) by adding new features and omitting optional features (stereotyped `<<o>>`). In Figure 2.1, the `ProductInstance` archetype has the optional attribute name and optional operations such as `getBatch()`. For archetype patterns, archetype relationships stereotyped `<<o>>` (e.g. `batch` in Figure 2.1) and classes stereotyped `<<archetype>>` are optional. These features provide points of controlled variation that can be "pruned" such that the resultant pattern remains semantically valid. "Hooks" are provided to add back the removed features.

Pleomorph is the variation of an archetype pattern for a specific business domain. Pleomorphism modifies the form of an archetype pattern such that the essential semantics of the base archetype pattern remain unchanged. The pattern may take on a different structure to suit the specific requirements of a business context (Figure 2.3). This may mean different archetypes, archetype features, and relationships in each of the variants.

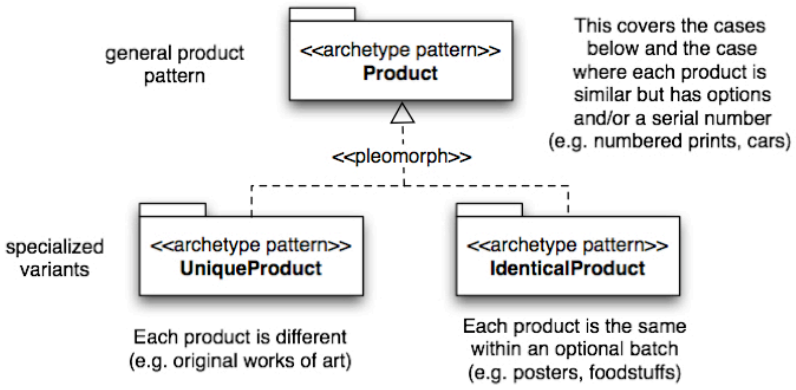

Figure 2.3: The `Product` archetype pattern pleomorphs [1]

2.3 Directives

Directives are elements of the business plan that define, constrain or liberate some aspects of an enterprise [22]. Directives are put in place by the business based on internal and external influences that can assist or hinder its operations. These include changes in the competitive, sociocultural, political or internal organisational environment [22]. Hence, Directives are volatile elements of the business plan. They govern how a business system should, or should not work in order to support enterprise activities. Directives are categorised as Business Policies and Business Rules.

Business Policies are Directives that provide broad governance or guidance to the enterprise. They define what can be done and what must not be done, and may indicate or set limits on how it should be done. They are non-actionable in the sense that their compliance cannot be determined from the observation of a relevant situation. An example Business Policy for a car rental company [22] is "Rental payments must be guaranteed in advance".

Business Rules provide specific and discrete governance or guidance to implement Business Policies. They are actionable such that a relevant situation

or behaviour can be observed and decided directly whether it is complying with the rule. Business Rules can be classified into Behavioural Rule and Structural Rule. The former governs the conduct of business activity while the latter defines how the business organises its business semantics. An example Business Rule based on the car rental Business Policy is "A provisional charge for the estimated cost of the Rental must be made against a valid credit card held by the Renter before the Car is handed over" [22]. Following the formal vocabulary in the OMG business modelling specification [16], Business Rules can be interpreted and used by computer systems to enable automation. The predefined linguistic structure enables natural language to represent and formally define requirements. Stakeholders can express their ideas in a language familiar to them in an unambiguous way.

3 Directive-oriented Pattern Analysis

3.1 Enduring and volatile requirements

In Directive-oriented Pattern Analysis (DoPA), requirements fall into two categories: enduring and volatile. Enduring requirements are stable requirements derived from core business activity while volatile requirements are likely to change, at any time [21]. Figure 3.1 illustrates the elements of DoPA and their relationships.

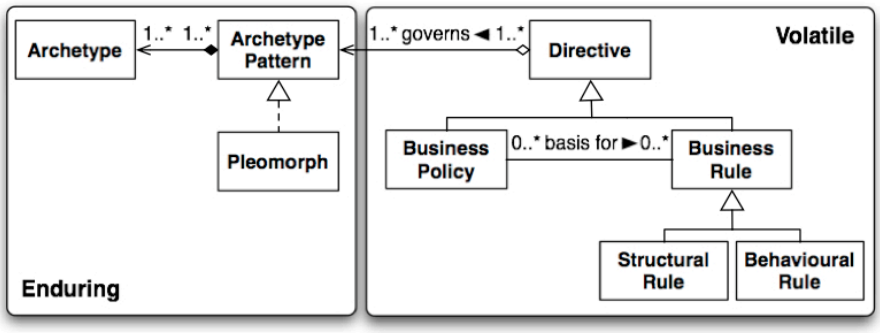

Figure 3.1: Elements of DoPA

Enduring requirements are associated with archetype patterns that are pervasive across business domains. DoPA uses archetype patterns with service choreography [19] as abstractions for common business workflows. To do so, the following information from the archetype pattern catalogue [1] are used: (1) functional requirements for common business needs captured as models; (2) rationale for using the pattern; (3) business context for the pattern; (4) stakeholders and their interactions with the pattern; and (5) pattern adaptation for a specific business context.

Archetype patterns are represented as parameterised collaborations of business processes to form workflows. Using this representation, instantiation of the pattern involves using services to play the roles of the placeholders. The services must accord with the semantic constraints of the pattern and its parameters.

DoPA uses archetype patterns as business service choreography patterns. The relationships between archetypes specify the interactions among processes in the workflow, similar to specifications for service choreography. Archetypes that are related via composition, association, and aggregation are parties in the interaction. The direction of the relationship defines the sequence of the messages. The multiplicity of the relationships defines the party responsible for the interaction, and parties allowed or required to send messages.

DoPA allows the combination of archetype patterns to assemble workflow abstractions. The following variations on archetype patterns are permitted: (1) dropping the `<<o>>` and `<<archetype>>` stereotypes on elements that are required by the business; (2) dropping optional features; and (3) structure variation via pleomorphism.

Volatile requirements are externalised as Directives. These include non-functional requirements [6] and business flow logic. Deriving Directives from volatile requirements ensures that system requirements are driven by the business and depicts changes in the business.

To relate Directives to SOC, the guidelines for their classification are as follows. Business Policies can be organisational or technological in nature. They include: regulatory compliance; conformance to standards and best practices; and corporate, divisional or departmental statements that provide directions for the enterprise. Business Rules are formulated based on Business Policies. They define or constrain a specific aspect of the business. Structural Rule constrains understanding of concepts specific to the business by built-in definitions. Behavioural Rule defines the behaviour required from the system to support business processes and workflows. These are related to business flow logic such as activities, sequencing, dependencies, and interactions.

Using the guidelines, Business Policies govern the configuration of workflows, while Business Rules guide the composition of services to form business processes. Hence, in a set of functionally equivalent services, Directives identify the one that best suits the business.

3.2 Managing requirements with DoPA

The DoPA approach focuses on requirements change and evolution of service-oriented business systems (Figure 3.2). The approach is adapted from the progression of steps in business planning for dealing with change [22]. Similar to [13], DoPA emphasises on managing change and not on requirements elicitation or conflict analysis. In Figure 3.2, the solid arrows indicate logical dependencies while the dotted ones indicate points of entry for changes.

Step 1: Identify requirements. This step involves the identification of the stakeholders and their needs from stakeholder interviews, business documents, and analysis outcomes [21]. The output is a list of requirements.

Step 2: Identify Business Policies. Business Policies are identified following the classification in Section 3.1. The source of Business Policies can be stakeholders, and business documents such as business plans. Broader Business Policies can be a basis for more specific Business Policies. The output is a list of Business Policies and dependencies.

Step 3: Select archetype pattern(s). This step selects pattern(s) from the archetype pattern catalogue [1] as workflow patterns to meet the requirements from Step 1. As DoPA focuses on commodity requirements via patterns,

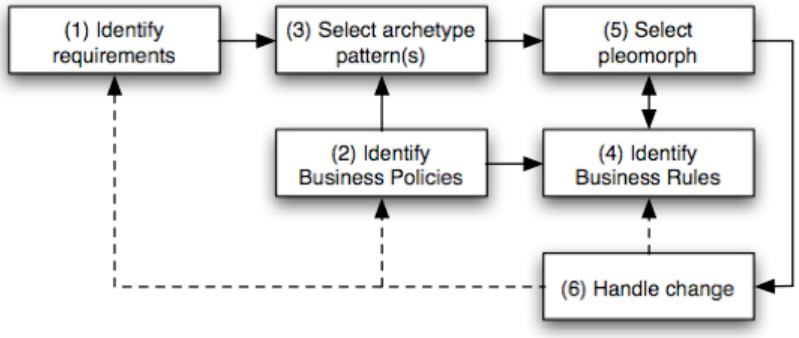

Figure 3.2: The DoPA approach

requirements for specialised business systems that cannot be mapped to the patterns are not handled. The Business Policies from Step 2 are used for workflow configuration. They introduce variations to the archetype patterns such as: (1) removal of the stereotype `<<o>>` in operations and relationships; and (2) removal of optional operations and relationships. The output is a service choreography specification with the configured variations.

Step 4: Identify Business Rules. The Business Rules identified here are based on the Business Policies from Step 2, following the classification in Section 3.1. Dependencies are identified to track the cascading effects on Business Rules when Business Policies change. The output is a list of Business Rules and dependencies.

Step 5: Select pleomorph. The archetype pattern(s) from Step 3 are applied to the specific business context by choosing the most appropriate pleomorph based on the Business Rules from Step 4. This can cause pleomorphism or schema variation to the archetype patterns. The output is a service composition specification that expresses the compositions required to meet the variations. This is a set of criteria to select business processes that meet the business needs by supporting workflow requirements while adhering to Business Rules. The business processes are composed services using building blocks (e.g. web services) that adhere to the choreography specification.

Step 6: Handle change. Evolution in the business system involves introduction, modification and removal of Directives and requirements. Changes in Business Policy maps to changes in service choreography specification, and is introduced into the system via configuration. Changes in Business Rules map to changes in service composition. The dependencies ensure that cascading effects of change are captured for traceability. In the cases where a Directive is removed, all corresponding dependencies need to be removed. Archetype patterns are well-defined, pervasive and related to core needs of the business. Hence, modifications to requirements that are associated with archetype patterns are not frequent, unless drastic changes occur in the business model. New Business Policies require reapplication of the approach from Step 2, while new Business Rules from Step 4. Finally, new requirements require the reapplication of the DoPA approach to select archetype patterns and identify Directives.

The DoPA approach produces criteria for selecting from various compositions provided by service providers. The ability to change among them supports evo-

lution of the business system. DoPA maintains service composition specification separate from service choreography specification. The former contains business flow logic and non-functional requirements while the latter contains only functional logic and abstractions of common service interaction. This provides a mechanism to ensure that the business processes composed from services can: (1) be integrated with the functional requirements from stakeholders; (2) be checked for adherence to the Directives; and (3) support workflow change via configuration changes. This provides a structured way to identify volatile requirements changes where service configuration and composition form the most important task to achieve the flexibility needed for agile systems.

4 Case study

This section validates the DoPA approach using a case study based on a Financial Information System (FIS) [24]. The FIS enables financial professionals of a fictitious Financial Consulting Company (FCC) to access real-time price, price-histories and statistics of stocks. FCC provides financial advice to its clients, who trade regional and industry focussed financial instruments. FCC requires a system that can supply financial information for a small and focussed client base. The market to which FCC offers advice depends on investor interest and market dynamics.

4.1 Step 1: Identify requirements

The stakeholders of the FIS are mainly financial analysts. They access market data and currency exchange rates to devise investment strategies. They require information on historical price and trading patterns to forecast price movements and identify buy and sell signals. To make financial projections, they also require historical and present stock data. The requirements are tabulated in Table 4.1.

Table 4.1: List of requirements for the FIS

ID	**Description**
R1	The analyst accesses market data and currency exchange rates.
R2	The analyst views statistics of historical price and trading pattern of stocks.
R3	The analyst checks current and historical stock price.

4.2 Step 2: Identify Business Policies

As FCC is a financial consulting firm, it does not engage in trading activities such as money exchange or stock trade. FCC attracts and retains customers by providing professional financial consulting to its clients. To do so, FCC focuses on consulting for specific markets depending on employee expertise. Currently, it concentrates on the analysis of the NASDAQ stock market. Finally, FCC is located in Australia, and targets Australian based investors. The list of Business Policies and dependencies are tabulated in Table 4.2.

Table 4.2: List of Business Policies for FCC

ID	Business Policy	Based on
P1	FCC provides professional financial consulting.	R1, R2, R3
P2	FCC does not engage in trading financial assets.	P1
P3	FCC provides advice on NASDAQ stocks.	P1
P4	FCC clients must be based in Australia.	P1

4.3 Step 3: Choose archetype pattern(s)

The requirements R1, R2 and R3 (Table 4.1) are concerned with stocks price and currency exchange rate. Stocks are goods that can be bought and sold. The `Product` archetype pattern provides a way to model goods and services. One of the archetypes is `ProductType` which describes the common properties of a set of goods or services. Foreign exchange is pervasive in dealings involving money. The `Money` [1] archetype pattern consists of the `CurrencyConverter` archetype which converts `Currency` from a source to a target by applying an `ExchangeRate`.

The Business Policy P2 (Table 4.2) ensures that FCC does not trade stocks. Hence, it does not require archetypes such as `Batch` (Section 2.2 Figure 2.1). The related operations are removed from the service choreography configuration. For the `Money` archetype pattern, the main references are made to the `CurrencyConverter` and `ExchangeRate` archetypes (Figure 4.1). The resulting service choreography specification consists of archetypes annotated with the required configurations (Table 4.3).

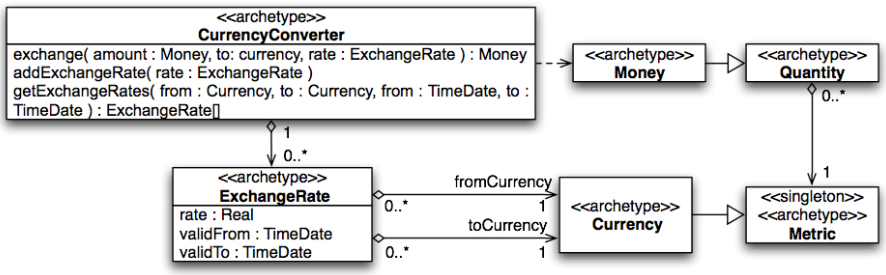

Figure 4.1: `CurrencyConverter` archetype [1]

Table 4.3: Partial service choreography specification

ID	Archetype Pattern	Service choreography requirements	Based on
CH1	`Product`	Prune archetype `Batch`	P1
CH2		`ProductCatalog`: NASDAQ	P3
CH3		`GetPrice context`: Last Price	P3, R3
CH4	`Money`	`toCurrency`: AUD	P4

[1] A sample chapter from [1] on the `Money` archetype pattern is available at http://www.codegeneration.net/articles/arlow/Arlow_ch11.pdf.

4.4 Step 4: Identify Business Rules

The Business Policy P1 (Table 4.2) emphasises on professional consulting services. To implement P1, FCC has the Business Rule that analysis submitted to customers must be backed up with statistics and graphs. Also, analysts must base their work on up-to-date market information. For convenient analysis, the present price of stocks is converted to local currency. FCC refers to this value as Local Last Price. The list of Business Rules and dependencies are tabulated in Table 4.4.

Table 4.4: List of Business Rules for FCC

ID	Business Rule	Based on
	Behavioural Rule	
BR1	Analysis must be backed up with statistics and graphs.	P1, P3
BR2	The financial information used for analysis must be updated every 20 minutes.	P1
	Structural Rule	
SR1	Local Last Price is the present price of a stock in local currency.	P1, P4

4.5 Step 5: Select pleomorph(s)

The `Product` archetype pattern (Section 2.2 Figure 2.3) can adapt into the `IdenticalProduct` and `UniqueProduct` pleomorph. As the purpose of the FIS is to monitor stock quotes, each stock is a unique product to FCC. Hence, the `UniqueProduct` pleomorph (Figure 4.2) is selected. If the requirements were for a stock trading system, the general `Product` archetype pattern would be used. This is because the stock quote will be for a `ProductType`, while the actual instance of stocks being purchased will be the `ProductInstance`. The `Money` archetype pattern does not require pleomorphism here.

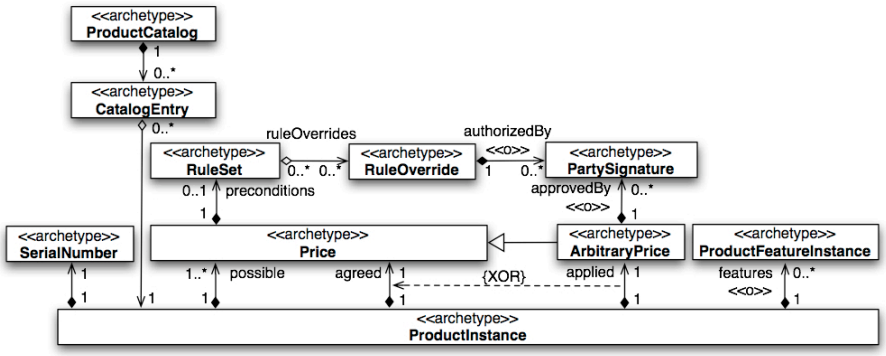

Figure 4.2: `UniqueProduct` pleomorph [1]

A service composition specification from the Business Rule BR1 is that annual historical quotes of Yahoo and Amazon must be compared to Google in a graph. Based on BR2, financial information must be updated every 20 minutes. SR1 results in the composition specification that the last price of stock quotes

must be displayed in Australian Dollar. Table 4.5 illustrates the contents of a service composition specification.

Table 4.5: Partial service composition specification

ID	Service composition requirements	Based on
CO1	Annual historical quotes of Yahoo and Amazon must be compared to Google in a graph.	BR1, BR2
CO2	Stock quotes must be updated every 20 minutes.	BR2
CO3	Last price of stock quotes must be displayed in Australian Dollar.	SR1

The output of the DoPA approach includes the list of requirements, Directives and specifications. The final step of the approach is a continuous process of handling requirements change. This step is demonstrated in the following section as part of the approach evaluation.

5 Evaluation

Evaluation is based on the criteria for evaluating RE approaches [21] which include managing change, mapping requirements to later stages, and traceability. Scalability, verification and validation are not addressed in this paper.

5.1 Managing change

Four typical business scenarios from [22] are applied to the FIS case to evaluate DoPA in supporting changes. The first two scenarios occur more frequently.

Scenario 1: The enterprise reacts to competition. FCC faces competition from another consulting firm and decides to get faster updates on its financial information, to provide more timely financial advice to its clients. The Business Rule BR2 (Table 4.4) is changed to: "The financial information used for analysis must be updated every 15 minutes". The service composition requirements CO1 and CO2 are affected as they are based on BR2. Original compositions that offer 20 minutes delay can no longer be used. This requires a change of compositions and/or service provider. The service choreography configurations are unaffected.

Scenario 2: The enterprise reacts to market change. Clients are shifting their interest to investments in Japan. FCC decides to change the market it operates in from NASDAQ to Nikkei. In this case, the Business Policy P3 (Table 4.2) is changed to: "FCC provides advice on Nikkei stocks". This requires a reconfiguration of the service choreography specification to access Nikkei stock quotes instead of NASDAQ. The Business Rule BR1 (Table 4.4), about displaying stocks in graphs, is based on P3. Hence, BR1 needs to be inspected for cascading changes, followed by the service composition specification CO1 (Table 4.5) which is based on BR1. CO1 is modified to include recomposition of Nikkei stock quotes into a graph.

Scenario 3: The enterprise diversifies. FCC decides to diversify its analysis to include providing advice in options and bonds. This requires the reapplication of the DoPA approach from Step 1. A new requirement R4 is introduced into the FIS: "The analyst accesses options and bonds commentary".

A new Business Policy formulated from R4 is P5: "FCC provides financial advice on options and bonds". The service choreography specification has to include configurations to access options and bonds information. This can also introduce new Business Rules to display the information. The `Product` archetype pattern is selected for composition as the information belongs to different product types, requiring the `ProductType` archetype.

Scenario 4: The enterprise expands. FCC decides to expand to Japan. This changes the Business Policy P4 to include Japan based customers. The service choreography specification needs to be reconfigured such that the `getExchangeRates` operation is set to convert to Japan Yen for the Japan based offices.

5.2 Mapping requirements to later stages

This section describes how the requirements derived from DoPA are mapped to elements in typical service-oriented architecture, design and implementation.

Service-oriented design and development. Service specifications in service-oriented design [18] contains three specification elements: (1) *structural specification* defines service types, messages, port types and operations; (2) *behavioural specification* defines service behaviour, effects and side effects of service operations; and (3) *policy specification* defines policy assertions and constraints on the service. The Directives in DoPA correspond to specification elements as follows. Business Policies correspond to *policy specification*. Structural Rules correspond to *structural specification* while Behavioural Rules to *behavioural specification*.

Architecture. The output of DoPA is mapped to the Service Component Architecture (SCA) component model [23]. SCA is a set of specifications that describe a model for construction, assembly and deployment of services to build applications using SOA. SCA divides the steps in building a service-oriented application into: the implementation of service components which provide and consume services; and the assembly of components to build business applications, through the "wiring" of service references to services (Figure 5.1). The mapping focuses on how business patterns and Directives define the "wires" between business logic components.

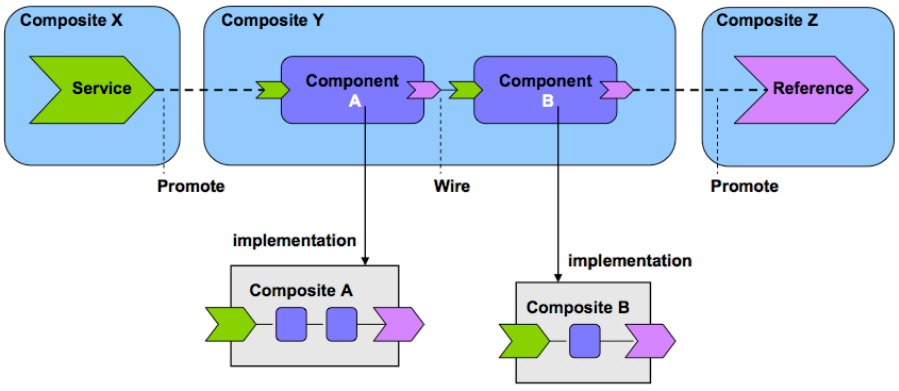

Figure 5.1: Service Component Architecture [23]

The SCA component type represents configurable aspects of an implementation. It consists of: services; references to other services that can be wired; and properties that can be set. These can be configured following the service choreography specification from DoPA, with archetype patterns governing the interaction between components. Service composition specification defines the "wiring" of service references to services such that the endpoints of a component map to their equivalent in the composition specification.

Implementation. A prototype Financial Information System is developed based on the specifications formulated from the DoPA approach. It uses the EditGrid [8] spreadsheet platform as an example domain specific language. This specific implementation can be generalised to other implementations as follows. Services such as exchange rates and stock quotes are configured from real-time remote data. Service composition maps to formulas performed on two or more remote data. Graphs can be composed from remote data or from existing compositions provided by vendors via remote image.

5.3 Traceability

Traceability is supported by guidelines and archetype patterns. With the DoPA approach and dependency tables, traceability is reversible, such that an enterprise can show why it has chosen a particular business process.

6 Related work

Existing RE approaches for services include: goal-oriented modelling of service capabilities (e.g. [20], [12]); service elicitation and orchestration requirements for new composite services (e.g. [11]); evaluating services based on their economic viability (e.g. [27]); and negotiations of conflicts between global and local business requirements (e.g. [26]). The main purpose of these approaches is not on managing change. They either focus on the requirements for new services, or assume the need to customise services for meeting individual business needs. In contrast to these approaches, DoPA aims to manage requirements for volatile systems, based on the concept of services commoditisation [4].

DoPA uses similar notion with product line architectures [7] which models a family of related applications as patterns and configures particular instances from the patterns. DoPA uses business archetype patterns making it specific for analysing business systems.

Business archetype patterns have been used as meta Platform Independent Models in Model Driven Architecture (MDA) [9], and as an ontology for business system messaging standards [1]. In MDA, non-functional requirements (NFRs) are realised by the executable model. Hence, only infrastructural NFRs are considered and not volatile business changes. Also, applying requirements change involves re-execution of the model. This results in replacing the entire system, which inevitably impacts on the business. DoPA departs from MDA by enabling changes to be made only to affected parts of the system. This is made possible by SOA and service utilisation.

The Service Mosaic framework uses state machines to model the behaviours of service choreography [2]. It provides the underlying work that enables DoPA to utilise services at the abstract level of archetype patterns.

7 Conclusion and future work

This paper proposes a novel approach for managing requirements of service-oriented business systems from the perspective of change and adaptability. The salient features of DoPA include: (1) utilisation of services with business archetype patterns as service choreography patterns; (2) mapping of business changes to service requirements via Directives; and (3) cascading of changes in Directives to service configuration and composition. The Financial Information System case study shows how DoPA models services to be configurable to meet volatile requirements while separating business flow logic as composition specifications. The evaluation shows how DoPA provides a structured approach for cascading changes in business requirements to service specifications. It provides a high-level mapping of the output of the DoPA approach to service-oriented architecture, design and implementation.

Future work involves addressing: (1) stakeholder dependencies and business intention analysis for eliciting requirements and Directives; and (2) alignment of changes in stakeholder dependencies and business intentions with service requirements.

Bibliography

[1] Jim Arlow and Ila Neustadt. *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML.* Addison-Wesley Professional, Boston, Massachusetts, 2004.

[2] Boualem Benatallah, Fabio Casati, Farouk Toumani, Julien Ponge, and Hamid Reza Motahari Nezhad. Service mosaic: A model-driven framework for web services life-cycle management. *IEEE Internet Computing*, 10(4):55–63, 2006.

[3] David Bush and Anthony Finkelstein. Requirements stability assessment using scenarios. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 23–32, 2003.

[4] Nicholas G. Carr. It doesn't matter. *Harvard Business Review*, 81(5):41–49, 2003.

[5] Betty H. C. Cheng and Joanne M. Atlee. Research directions in requirements engineering. In *Proc. of Int. Conf. on Soft. Eng. (ICSE)*, pages 285–303, 2007.

[6] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-functional requirements in Software Engineering.* Kluwer Academic Publishers, 1999.

[7] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley Pro., 2002.

[8] EditGrid. *www.editgrid.com.*

[9] David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing.* Wiley, 2003.

[10] Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[11] Rim Samia Kaabi, Carine Souveyet, and Colette Rolland. Eliciting service composition in a goal driven manner. In *International Conference on Service Oriented Computing (ICSOC)*, pages 308 – 315, 2004.

[12] Lin Liu, Chi-hung Chi, Zhi Jin, and Eric Yu. Strategic capability modelling of services. In L. Baresi, X. Franch, and N. Maiden, editors, *Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'06 - RE'06 Workshop)*, page 3, 2006.

[13] Ana Moreira, Joao Araujo, and Jon Whittle. Modeling volatile concerns as aspects. *CAiSE 2006. LNCS*, 4001:544–558, 2006.

[14] Chris Nott. Patterns: Using business service choreography in conjunction with an enterprise service bus. *IBM Redbooks paper*, page 32, 2004.

[15] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *Proc. of Int. Conf. on Soft. Eng. (ICSE)*, pages 35–46, 2000.

[16] Semantics of Business Vocabulary and Business Rules (SBVR) Specification. *in OMG, www.omg.org*, 2006.

[17] Michael P. Papazoglou and Dimitrios Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.

[18] Michael P. Papazoglou and Willem-Jan Van Den Heuvel. Service-oriented design and development methodology. *Int. J. of Web Eng. and Tech. (IJWET)*, 2(4):412 – 442, 2006.

[19] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, 2003.

[20] Loris Penserini, Anna Perini, Angelo Susi, and John Mylopoulos. From stakeholder intentions to software agent implementations. *CAiSE 2006. LNCS*, 4001:465–479, 2006.

[21] Ian Sommerville. *Software Engineering*. Addison-Wesley, 7th edition, 2004.

[22] Business Motivation Model (BMM) Specification. *version 1.0, in OMG, www.omg.org*, 2007.

[23] Service Component Architecture Specifications. *in Open Service Oriented Architecture, www.osoa.org*.

[24] Financial Information Systems. *in Reuters, http://about.reuters.com/productinfo/s*, 2008.

[25] Dave Thomas. Next generation it - computing in the cloud life after jurassic oo middleware. *Journal of Object Technology*, 7(1):27–33, 2008.

[26] Paolo Traverso, Marco Pistore, Marco Roveri, Annapaola Marconi, Raman Kazhamiakin, Pierluigi Lucchese, Paolo Busetta, and Piergiorgio Berto. Supporting the negotiation between global and local business requirements in service oriented development. In *Int. Conf. on Service-Oriented Computing (ICSOC)*, page 10, New York, USA, 2004.

[27] Bas van der Raadt, Jaap Gordijn, and Eric Yu. Exploring web services from a business value perspective. In *Proc. of the IEEE Int. Req. Eng. Conf. (RE)*, pages 53– 62, 2005.