Identifying functional variants of Publish/Subscribe systems for adaptive components of Medium

Anthony Lee Ka Chun

Internship report on:

Identifying functional variants of Publish/Subscribe systems for adaptive components of Medium

Research Master's Degree in Computer Science 2009 – 2010
August 13th 2010

Lee Ka Chun Anthony
Advisors of CAMA: Maria-Teresa Segarra, Jean-Marie Gilliot

Table of Contents

Abstract:

Component adaptation is one processing approach to develop flexible, reusable, traceable and reliable applications in modern software engineering industry [19]. When applying this technology to Publish/Subscribe paradigm for its different functionalities, there are several aspects such as using methodology and variants identification have to be considered. The principle of Model-driven architecture is used to construct the adaptive architecture of Publish/Subscribe paradigm because of its advantages [22, 24]. It could be used to classify different system aspects into models and the components adaptation could be performed by the transformations of these models. In order to identify the functional variants of Publish/Subscribe systems for establishing these models, fundamental researches on some famous Publish/Subscribe systems are necessary. The variations should be identified carefully base on the researches because they will affect any further developments according to their closely linked relationship. All these processes were finished in this internship and a methodology is successful established with the proofs of a prototype. The details of each aspect are analyzed in this report.

1. Introduction

The background and overview of the research internship are provided in this section. The development approach and steps are also introduced in the following paragraphs.

1.1 Background

Publish and Subscribe (Pub/Sub) paradigm is one of the popular application architectures in distributed systems. It particularly focuses on the problem of coupling, redundant message and user addressing in a large scale system [1]. Is is a loosely coupled communication paradigm that cut off the direct connection between senders (publishers) and receivers (subscribers) [10]. Publishers and subscribers are decoupled in space, time and flow [1]. This nature highly increases the scalability and efficiency of a messaging system. It also helps to avoid failure message delivery between source and destination caused by disconnected or blocked nodes.

There are many researches proposed different functionalities and characteristics that could be used to implement this paradigm [28, 31, 33, 36]. However, these functionalities and characteristics may not be able to operate concurrently in a system because they were designed separately for different purposes. Also, these variations are always changeable to satisfy different user requirements which means that re-implementations or re-installations may required frequently, and both of the approaches are costly and time consuming [8]. In order to avoid these costly approaches, component adaptation should be considered during the application development cycle.

Component architecture adaptation is a type of dynamic adaptations [4]. Dynamic adaptation is a software architecture concept for benefiting the design and programming aspects of an application [4]. The main idea of software adaptation is to increase the ability of softwares for modifying the components not only during the implementation period, but even after the software has been built and deployed. When it is applied to an application, the application is able to modify its behaviors such as function parameters, locations and structures of the application for achieving a changeable execution context. The context could be environmental executions or user activities. When these modifications can be performed without any service interruption, adaptations can be considered as dynamic [3]. Component architecture adaptation separates system functionalities (internal structures) into different components to avoid redundant implementations. Sets of variants of components should be defined carefully for abstractive model constructions. Once the constructions are finished, components can be added, removed or replaced [4, 8]. Therefore, different functionalities of Pub/Sub systems could be obtained quickly and easily when an adaptive solution is applied.

In order to realize component architecture adaptation to publish/subscribe paradigm in this internship, the concept of Model-Driven Architecture (MDA) is used in the development process [3]. It is an software engineering approach aimed to create a software design model for software developments on different platforms [24]. It separates development processes into different layers and each layer is corresponding to specified model. The importance of MDA is to construct an application that can be ran under different operating systems by just requiring model mappings and transformations to generate different kinds of source code, instead of duplicated implementations. Although MDA is not specifically for solving

functionality problems, this research takes advantage of this idea to create series of models for implementing Pub/Sub systems. The idea is using the mapping technologies as defined variants with specified functionalities so that each variation of Pub/Sub system could be transformed and generated easily by just changing a mapping model.

1.2 Internship direction

The purpose of this research is to identify the functional variants of Pub/Sub systems in order to establish different models for variant transformations. The transformations are aimed to create a final model that contain specified configurations, which can be used for actual source code generation. A prototype should be constructed to simulate this development process and to prove the solution concept. When there are lack of researches on combining component adaptation and model-driven architecture as a development solution for Pub/Sub paradigm [28, 31, 33, 36], the challenge of this research is to identify suitable variation points (which related to specific functionalities) for building efficient models. It can directly affect the quality of outcome (usability of the generated source codes) if the variation points are not accurate enough. Therefore, further exploration of famous Pub/Sub systems is necessary in the internship.

The internship was divided into 3 main steps. First, identify functional variation points of Pub/Sub systems by exploring several famous existing systems. When variations points are identified, establish the variant models for transformation processes. Finally, adopt these models to a prototype for implementing the solution. JAVA and Kermeta [26] programming language are used to implement the prototype with an interaction abstraction concept which called “Medium”. It is a middleware architecture proposed by E. Cariou and al [21]. Its details will be mentioned in 2.3.

The paper is organized in 6 sections including the introduction and each section provides explanation on each step. Section 2 states the further research on Pub/Sub systems and give analysis on each systems and its functionalities. In section 3, functional variation points of Pub/Sub paradigm and their models are analyzed. The principal contribution of this research is stated in this section. The experiments of the prototype is mentioned in section 4. It occupied the major workload of the internship. The prospectives and the conclusion of this research internship are stated in section 5 and 6 respectively.

2. Further research on Publish/Subscribe systems

In this section, the further exploration of existing famous Pub/Sub systems is analyzed to understand the main variation points. Then, combining these analysis with the viewpoints of some significant researches [1,2,6], the initial table of the variants is obtained by the research results

2.1 Exploration of the famous Pub/Sub systems

The basic interaction scheme and concept of Pub/Sub paradigm were explored and analyzed in previous paper [40]. There are several crucial ideas that can be summarized.

- Pub/Sub paradigm is generally combined by 3 main components, which are Publishers, Subscribers and Service Brokers.

- Service Brokers provide the main functionalities "Publish" to the Publishers, "Subscribe", "Unsubscribe" and "Notify" to the Subscribers. This communication scheme decoupled the direct connection between Publishers and Subscribers which allows more flexible services. Thus, the service broker could be viewed as "one-side" service for both of them [6].

- Most of the variants are related to the implementation of the Service Brokers such as filtering methods, notification policies and operation conditions.

The general Pub/Sub paradigm is depicted by these ideas. However, these ideas are very abstract to compose the functional variants that we need. More evidences are required to categorize accurate variation points. In order to identify relevant variation points, 5 existing famous Pub/Sub systems were explored to understand their system architectures and the areas that they emphasized and interested. These 5 systems are Le Subscribe, Scribe, Gryphon, Siena and Hermes. Their crucial ideas are stated in the following paragraphs.

Le Subscribe

Le Subscribe is a Pub/Sub system developed by INRIA [33]. The purpose of the system is to provide efficient matching services in large-scale Web-based environment, specially deal with highly dynamic Web information. The system supports user input predicates for filtering processes in a LDAP liked hierarchy. This specified hierarchy is aimed to distinguish different objects in different domains in order to enhance event rising and searching performance. Subscription language is a combination of attribute name, comparison operator and value. Events have to satisfy all requirements of a subscription in order to be distributed to subscribers. This well formed structure enhanced the expressiveness of the information. Together with predicate based algorithm to optimize redundancy and dependency among subscriptions, the searching and matching speed of the system would be fast and smooth because the number of predicate evaluation is reduced [32]. It also provides different publication and subscription interfaces such as event polling and email delivery to strengthen the capabilities of the system. This research system is a content-based Pub/Sub paradigm that mainly interested in performance speed and content hierarchy.

Scribe

Based on Pastry Pub/Sub system, Scribe is an extended system developed by Microsoft research laboratory. It is a decentralized Pub/Sub model aimed to provide routing management and content management [37]. By its topic-based feature, the system allows subscription on publisher customized topics which means that content management is flexible. It provides best-effort dissemination message delivery and subscriber management mechanisms by using the peer to peer overlay network infrastructure of Pastry with own implementation of multiple brokers on multicast network. Subscribers are connected to related brokers to form different multicast trees so that events can be broadcast to correct destinations after they were delivered to target brokers. The multicast trees are well balanced by the system and its performance is efficient. However, the drawback is message order cannot be guaranteed [36]. The systems also supports recovering for fault-tolerance mechanism. It mainly concerns the routing performance and delivery latency.

Siena

A content-based Pub/Sub solution proposed by University dell' Aquila and University of Colorado at Boulder [35]. It is aimed to provides wide-area event notification services that suitable for supporting highly distributed applications that requiring component interactions ranging in granularity from fine to coarse. This system can be viewed from 2 sides. On one side, it is a Pub/Sub system that performs specialized routing on application level network by using 2 forwarding tables with topological constraints and selection predicates, which filtering messages by IP address and subscriber requirements. Its routing algorithms enable multiple brokers communication for effective message dissemination. The expressiveness of filters, redundant filter propagations and message routing are concerned by this side. On the other side, it is a Pub/Sub middleware aimed to provides aspect controls for dynamic reconfigurations [34]. In order to create a framework that provides dynamic manage performance attributes, 3 steps are performed for the implementation which are application monitoring, model-based performance evaluation and dynamic reconfiguration. The system is emphasis on reconfiguring system inter-networking by concerning the performance aspect on this side.

Gryphon

Gryphon is a Content-based Pub/Sub system that proposed by IBM research laboratory. It is structured as a redundant overlay network that aimed to provides services with scalability, availability and security [29]. The system is supporting for application level network and multiple operating systems. It treats events as events and they are described via an information flow graph which specifies event delivery, event transformation and derived event generation [28]. It allows system management with event transformation and event stream interpretation which are 2 features offer function projection to the data in events and collapse/expand sequences of events respectively. The core of the system is using a patented matching engine to provide high-speed content filtering and it organizes topics into hierarchies [28]. The system guarantees message ordering and message validating with the controls of their fault-tolerance mechanism. As described in their researches, the system is

emphasis on event handling, content matching and delivery handling.

Hermes

A type-based Pub/Sub system that provides services on logical network of self-organized event brokers which was developed by University of Cambridge [31]. The system is emphasis on scalable routing and access controls. Each node (Publisher or Subscriber) in the system has a unique random numerical identifier which connected on a peer-to-peer overlay routing network. Subscriptions and publications are associated with specified event type for runtime type-checking. Those event types are organized into hierarchies and inheritance to create more specialized types. It is focusing on clean programming language integration to reduce the complexity of designing and building large-scale distributed systems. It is also concerning the security issues in terms of access controls by strongly typed roles for publishers and subscribers [30]. Furthermore, data accesses by querying are enforced with restrictions to increase information security.

After reviewed these 5 famous Pub/Sub systems, their characteristics and interested research areas could be summarized in the following table.

Systems	**Characteristics**	**Interested areas**
Le Subscribe	Predicate filtering, LDAP like information hierarchy, Efficient matching algorithms	Performance in large-scale web environment, Information matching speed and methodology, User connection interface
Scribe	Decentralized model, Multiple brokers implementation, Multicast routing	Routing algorithms and their performance, Network infrastructure, Delivery latency
Siena	Expressive predicate filtering, Routing table for multiple brokers, Dynamic reconfiguration	Routing and filter performance, Delivery methodology, System architecture
Gryphon	Information flow graph for events, Guarantee message ordering and message validating, high speed content filtering	Event handling, Delivery performance, Filtering performance, Fault-tolerance
Hermes	Connection access controls, Event type-checking, Filtering query restrictions	Information security, Access security, Filtering performance

Table 1. Characteristics summary of 5 famous Pub/Sub systems

As we can see in summaries, there are different concerning points of each system. Nevertheless, their interested areas are quite consistent. Most of them are focusing on the performance of routing algorithms and matching algorithms as these are the competitive areas of pub/sub paradigm. Almost none of them explicitly mention on the details of the basic interaction scheme of Pub/Sub paradigm. This is because current researches on pub/sub paradigm are more willing to focus on advance technique areas rather than generate interaction scheme that did not affect much on performance.

Although the functional variation points cannot be formalized here, the global direction of interested areas of current researches are generalized. They are including routing performance, delivery performance, filter performance, event handling, security issues, user interfaces, etc. These areas inspired the categories of the functional variants of Pub/Sub paradigm. The information in this section is not intent to provide an in-depth state of the art of current Pub/Sub systems, but rather to retrieve the global point of view from their research directions in order to consolidate our variation categories. Therefore, it is irrelevant to explore all exiting Pub/Sub systems as they are too many but without significant differences. The details of their technical implementations are not stated and considered also because they are out of the scope of our identification process. However, give each system an essential description is necessary to point our their global viewpoints.

2.2 Identify Pub/Sub system variants

According to the analysis of 2.1, certain variation categories are obtained. But in terms of concrete variation points of Pub/Sub paradigm, they are still quite ambiguous. Therefore, they should be combined with the viewpoints of some famous research papers. P. Eugster et al.[1, 2], one of the famous research papers that describe Pub/Sub systems from various angles of view, they summarized the characteristics and variants of current Pub/Sub systems into 4 categories:

Categories	**Basic Interaction scheme:**	**Alternative communication paradigm:**
Contents	Delivery mechanisms (pull/push, aperiodic/periodic), Functions, (subscribe, subscribe, publish, advertise), Decoupling requirements	Message passing, Remote invocations (e.g. fire and forget), Notifications, Shared spaces, Message queuing
Categories	**Pub/Sub variants:**	**Implementation issues:**
Contents	Topic-based, Content-based, Type-based	Media (centralized, distributed) Communication mechanisms (peer to peer, IP multicast), Quality of service (Reliability, priorities, security)

Table 2. Characteristics and variants summary of P. Eugster et al [1, 2] researches

As shown in the table, the functionalities and characteristics of Pub/Sub paradigm are more clear and specific. For example, delivery mechanisms such as periodic or push are categorized as basic interaction scheme. The Pub/Sub variants they classified as Topic-based, Content-based and type-based which could be interpreted as the natures of filtering. Some technical usages such as message queuing and remote invocations are viewed as alternative communication paradigm. When referring these elements to the summaries in 2.1, delivery performance and its variations could be considered as an item under basic interaction scheme and alternative communication categories. Filtering performance and its variations could be considered as an item under Pub/Sub variants category. Event handling and security issues could be considered as an item under alternative communication paradigm and implementation issues categories. Although there is no direct inheritance between both analysis, the connections between them could be viewed as fusions.

The importance of the explorations in 2.1 and 2.2 is to withdraw the concrete variation points and their categories from a very abstract scheme. They provide enough evidences to establish the corresponding elements. In consequence, we obtained the initial table (table 2) of Pub/Sub variants. In order to complete the first step of this research, the relevant functional variants should be identified. Considering the elements in table 2, they are concrete enough but the categories are too wide and some of the contents are not specific to our solution scope. For example, IP multicast of communication mechanisms under implementation issues is indeed a type of internal technology usages which is not suitable be a functional variant. Therefore, it is necessary to narrow down the width of those elements and finalize the functional variants. However, the core of the research and the solution scope should be emphasized before the finalization.

2.3 Adaptive medium and the solution scope

To effectively select the variants that we need, the scope of the solution should be considered. The Pub/Sub model of this research is built on top of a middleware logic called “Medium” (or Interaction abstraction). It is a generic middleware model that aimed to provide different functionalities by separating them into components. The purpose of this research is to adopt the functional variants that we identified to this architecture [41].

A distributed application involves many internal interactions and communications. They are always related to the major functionalities of the system. Software components are the reifications of them [21]. E. Cariou and al. [21] summarized the characteristics and the main properties of software components in order to introduce a concept as Medium. It defines a software component is an autonomous and deployable software entity and point out that it should clearly specifies the offered and required services. Thus, it could be used by any parties without understanding the internal logic such as algorithms. It should be capable to combine with other components also in terms of reusability and flexibility [21]. This definition is aimed to clarify the responsibilities and the data that has to be managed by software components in order to raise the traceability between design and implementation stages.

When building a Pub/Sub system that using Medium as the concept base, the functional requirements and non-functional requirements are considered as individual components that could be adapted or swapped during any time of the development cycle. Thus, the system could be evolved by dynamically add, drop or modify the components. This architecture concept could be visualized as Figure 1.

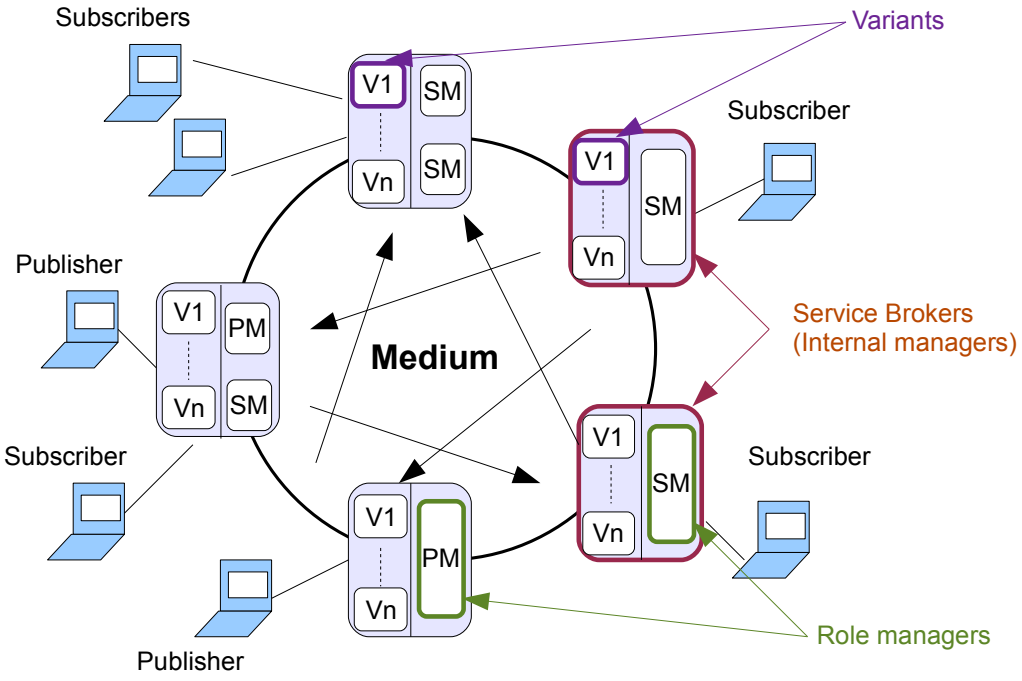

Figure 1. Pub/Sub paradigm built on top of Medium architecture with adaptive components

The concept of Pub/Sub Medium is described in Figure 1 which including all Service Brokers, Role managers, variants and the communications. When a client computer connect to the Medium, a related role manager is offered to it for operating any interaction between them. Subscriber in this case will be offered a Subscriber Manager (SM) and Publisher will be offered a Publisher Manager (PM) relatively. The Service Brokers shown in the figure could be imagined as Medium internal managers which contain core functions of the system. They are the instances of several structured functional skeletons. These brokers are supposed to supervise multiple role managers with their own client. One important concept here is that these distributed brokers could also be viewed as an unified entity because the Medium architecture is presented as a global unit that provides platforms to the users, which means that its internal architecture should be hidden from users.

The variants here are described as components of the brokers which are all adaptable. The abbreviation V1 to Vn is representing the specified variants held by the internal managers. “n” is the number of variation points (or categories). As they are implemented from the same skeleton, the types and the numbers of variants should be the same for each brokers. It should be emphasized that these variants are including the functional variants that needed to be

identified in this research and also including the non-functional variants that established before. By implanting this idea, our Pub/Sub system is not only maintaining the original purposes such as scalable for publishers and subscribers, but also carrying flexible characteristic for enhancements and maintenances because of the adaptive variants.

In this section 2, the basic interaction schemes of Pub/Sub paradigm is stated to understand the abstraction of the variation points. Then, further exploration of existing famous Pub/Sub systems helps to consolidate the variation points. The initial table of the variants is obtained by combining all research results. However, some of the elements are out of scope of the research and the concept of the solution is needed to be verified. After investigated all the materials, the functional variants should ready to be finalized in next section. All the analysis in this section are aimed to deliver a fundamental support for the first step of our research.

3. Contribution: Modeling functional variants of P/S systems

The entire development approach for demonstrate the research result is presented in this section. Together with different explanations of conceptual diagrams, the details of each aspect such as variant modeling, component adaptation and model-driven architecture will also be mentioned below.

3.1 Finalize the functional variants

By looking at the categories in table 1 and table 2, the definition of functional variant is too widely and they did not declare the functional and non-functional aspects clearly. Especially these aspects may cared by different users. For example, routing algorithms and their performances that summarized in table 1 are concerned by end-product developers as they want to have better quality products to sell. However, most of the end-users of these products probably don't want to understand the algorithm logics behind the systems because they just concern the product is easy to use or not, fast to get the result or not. In table 2, there are only 3 keywords are named variations of pub/sub paradigm which are Topic-based, Content-based and Type-Based. They are considered as variations of pub/sub from the nature point of view of the whole system. Other characteristics and functionalities are defined as scheme, mechanism or paradigm, QoS even belong to implementation issues. These definitions are not exactly what we want to find out as variation points and variants because they are still separated in different areas without some clear guildlines. In order to narrow down those elements, the word "Functional" need to be clarified in this finalization because it affects our decision significantly.

3.1.1 Functional and Non-functional requirements of software applications

Depending on different angles of views, any modifications for fulfilling user requirements are all affecting the "functional" aspect a system. The goal of our research is to identify the "functional" variation points of general Pub/Sub systems in order to modeling them for adaptive transformation. The word "functional" could be a bit tricky here, it may bring different representations to different layers of users and these representations may not be interchangeable between each layer [39]. The layers of users in our case are including middleware developers, end-product developers and end-users. It is necessary to clarify which layer of users and what suitable representation of "functional" that should be cared in this research. The viewpoints of functional requirements and non-functional requirements in engineering communities have to be classified also. However, consider the following example stated by M. Glinz, it is still arguable to classifying functional and non-functional requirements of a system in the communities [38]:

A particular security requirement could be expressed as:

1. "The system shall prevent any unauthorized access to the customer data"
   = **non-functional requirement**

Furthermore, it could be presented more specifically as:

2. "The probability for successful access to the customer data by an unauthorized person shall

be smaller than 10^{-5} " = **still non-functional requirement**

However, its original idea could be refined as:

3. "The database shall grant access to the customer data only to those users that have been authorized by their user name and password" = **functional requirement**

According to the definitions found in the article [38] that including IEEE committee definitions, expression 1 and expression 2 should be classified as non-functional requirements. But, when the semantic is refined to expression 3, it should be classified as a functional requirement. These sentences are representing the same requirement but with different expressions, they affect the classification between functional and non-functional categories. This example is aiming to point out expressiveness is an important factor that affects the identification of a requirement. Moreover, M.Glinz [38] provided various kinds of evidences and statements to propose there are 3 main problems causing the identification arguments, which are definition problems, classification problems and representation problems. And he suggested some new terms such as performance requirements, specific quality requirements and constraints which attempt to solve these problems. Unfortunately, his new methodology is requiring a consensus from the community by a large quantity of usage.

Nevertheless, some of the essential ideas of his suggestions are useful for finalizing the functional variants in this research. The concept of concern and constraint in his suggestion [38] are retrieved for establishing the final table of variants.

3.1.2 Establish functional variants of Pub/Sub systems with constraints

By considering constraints with concerns as the guildlines for final variation points and functional variants, a user group need to be targeted first. As a middleware type application, the concerned users will be those who use the services which are end-users and interface developers. "**End-users"** in this case which means the people who directly use the services provide by the application like publication and subscription. The interface developers are considered as our end-users also because different interface implementations are not affecting the services provided by the application but they are just affecting the services they provided to their end-user. Consequently, all other types of users are excluded by this setup. Second, instead of using the words such as mechanisms, interaction, scheme, functions, etc, the term "**behavior**" is used to clarify the nature of the selected elements. It is because the semantic of this word is closer to functionalities and also closer to the concerns of end-user. Together with these concerns, the constraints for the finalization are defined as following:

Constraint definition 1: <u>Visible to End-users</u>
The finalized behaviors should interested by end-users to know but they should not be required to have extra knowledge on how these behaviors works. E.g. End-users need to know the message notifications are in order or not, but they do not need to know the ordering behavior is performed by message queuing or some other technologies.

Constraint definition 2: Provide to End-users
The finalized behaviors should be provided as services to end-users according to the purpose of the application. E.g.: The system allows predicates such as SQL for filtering messages, or the system allows automatic filtering by predefined roles without any filtering languages.

Constraint definition 3: Concerned by end-users
The finalized behaviors should be concerned by end-users for their adjustments of the product, which could be expressed as those behaviors they would like to change periodically. E.g.: End-user may consider to change the access behavior of the system from open connection to authentication, or they may want to change the authentication type from password-based to identity-based to increase the quality of security.

Constraint definition 4: Exclusive in nature
The finalized behaviors should be logically exclusive by their nature which means that they could not be performed at the same time. This is an important constraint that make the functional variants meaningful because it restricted that these variants could not be implement together according to their architectural differences, and that's why they should be identified as changeable variants. E.g.: Message notification is either periodic or instant but could not be performed both at the same time logically. When changing message notification behavior from instant to periodic, the implementation architecture have to change to support message buffering. This may cause re-architecture implementation of the system.

According these constraints, the "functional variants" of Pub/Sub paradigm based on the previous researches could be finalized into 4 categories (variation points) as below:

Delivery behaviors	**Event behaviors**
(Push, Notify / Pull, Request) (Periodic / Instant) (Guaranteed / Best-effort)	(Ordered / Causal) (Storage / Fire and forget)

Filter behaviors	**Access behaviors**
(Predicate / Automatic) (Content-based / Topic-based / Type-based)	(Anonymous / Authentication) (Single access / Multiple access)

Table 3. Finalized functional variants of Pub/Sub paradigm in 4 categories

The categories are named as delivery behaviors, event behaviors, filter behavior and access behaviors. The meanings of the categories and their variants are quite straightforward. Delivery behaviors are concerning the message delivery issues for subscribers, event behaviors are concerning any types of event raised by provided services "Publish", "Subscribe" and "Unsubscribe". The variants under event behaviors here are emphasizing on publication events such that those event could be stored (Storage) or dropped (Fire and forget) after disseminated. Filter behaviors are concerning different types of matching between

publications and subscriptions that allowed by the system. The nature of access behaviors here is more special than the others because it is actually under security issues. However, a category named as security behaviors would be too widely when considering our scope and constraints. For instance, some hacking prevented implementations could be included also if it named so. When only concerning the variants of information accessing issues of end-users, the name access behaviors is more suitable for the our scope. This classification of categories (variant points) is aiming to provide generic spaces for specific elements (different variants).

The variants under each categories are identified according to those 4 constraints. They may not need to satisfy all 4 constraints but at least one constraint has to be satisfied. There are different representations of the horizontal allocation and vertical allocation under each category. The variants allocated horizontally with a slash "/" are opposite implementations that cannot be operated concurrently. For example, publication events can either be ordered or in-ordered (Casual) only, but not both. A comma ","means variants have a common style and they represent a same concept. Variants allocated vertically with brackets "()" are combinable implementations that could be operated concurrently. For example, message delivery could be implemented as periodically notify, instantly notify, periodically request or instantly request, but cannot be implemented as notify and pull nor periodically and instantly, which are not logical. Another example for this could be different behaviors of access. The system can be implemented as open access (anonymous) and allows multiple connections for 1 user, or require authentication and only allows single connection for 1 user, but not cannot be implemented as single and multiple accesses for 1 user, which not logical. These characteristics are applicable to all 4 categories. Therefore, a combination including ordered and storage or solo implementation of ordered for event behaviors are also possible.

It should be known that when combining the variants vertically, the number of the possibility are increased by multiplication. Also, the variants under each category should not affect each others. For example, changing the filter behaviors from topic-based to type-based should not impact any fixed event behaviors. Noticeably, the functional variants are not limited as shown in the table. Additional variants are possible to join under each category as long as their natures are suitable for that category and fulfill any of the constraints.

When defining the variation points and the variants, it seems that routing algorithms, matching algorithms (not meaning for matching type but different implementations for the same type of matching that concerning their performances), user interface implementations and reliability issues should also be considered. However, they are ignored in this table. Although routing algorithms and matching algorithms are main functionalities of Pub/Sub paradigm, they should be ignored if we follow the "concerned by end-users" constraint listed above. The reason is that end-users probably do not willing to understand how those algorithms work because they are not supposed to be experts in such areas. Similarly for user interface implementations, they are not the services that aim to be provided referring to "provide to end-users" constraint. Reliability issues are considered in all aspects rather than be separated into individual category. In this research, we rather to classify it as non-functional variation than functional variation. However, they are still adaptable as variants under these categories if they are needed by any further developments.

Step 1 of the internship is completed until this section. It is necessary to spend lengthly analysis and explanation on this step because it is the first and most important step of the processes. There would be serious deviations for all other steps if the functional variation points and the variants are not identified precisely. They are now ready for modeling.

3.2 Modeling of the variants

The final definition of the functional variation points and the variants are identified in previous section. In order to achieve the final solution, they must be transformed into models and metamodels. As mentioned in section 2.3, the functional variants are used to adopt on the top of a base architecture named "Medium". Medium is a generic middleware architecture aiming to be applied with different implementation paradigms. The Pub/Sub paradigm was the one chosen in our researches [41]. There are various in-depth researches for this generic software architecture [3, 21, 41], but they are not going to be analyzed in this paper as it will be out of scope of our research. However, the global concept of it has to be clarified for understanding "why" and "how" the functional variation points and the variants are going to be model. As the implementation architecture of Pub/Sub medium is analyzed in section 2.3, its conceptual architecture will be analyzed in this section.

3.2.1 Component adaptation with models

At first, 2 important metamodels have to be introduced which are Medium definition metamodel and Medium implementation specification metamodel. They are the initial system metamodel and completed system metamodel respectively.

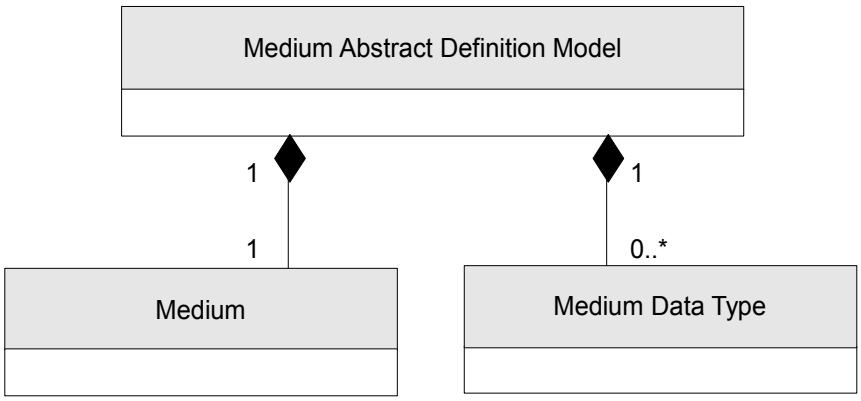

Figure 2. Medium definition metamodel class diagram

Figure 2 shows the class diagram of Medium definition metamodel. It needs to be reminded that metamodel is the structural rule of an actual model which could be expressed as the model of models [23]. It means that an actual model should be established according to its definition. This metamodel is relatively simple when comparing with the completed metamodel in figure 3. There are 3 classes contained in this diagram which are Medium abstract definition model, Medium and Medium data type. The Medium abstract definition model is the actual model class that could be instantiated for the initial transformation.

Medium is a class that contains all the initialize information about the Medium such as types and numbers of role, data resources and so on. It should be referred to the abstract definition model and should be referenced by it once only. Medium data type is a class for referencing different primitive data types such as integer, string, boolean of an actual programming language, for example JAVA. All these primitive types should inherit from the medium data type in order to be held by the abstract definition model as fundamental elements.

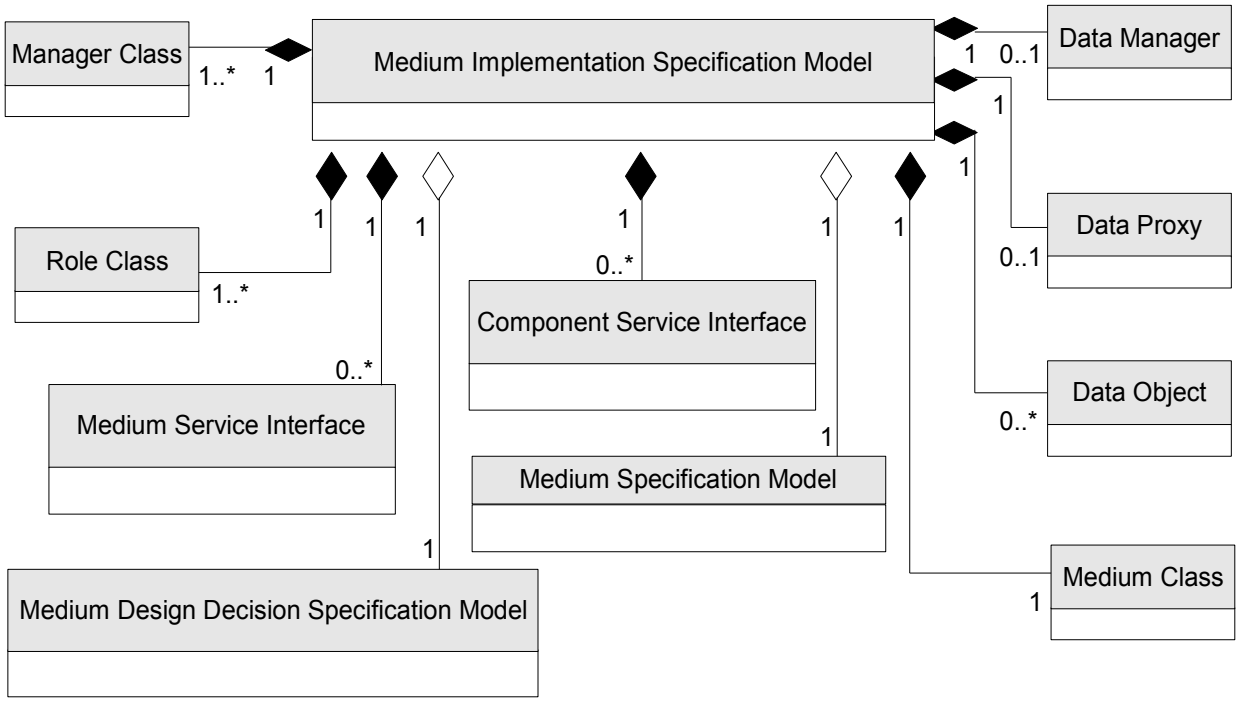

Figure 3. Medium implementation specification metamodel class diagram

Figure 3 is the completed system metamodel transformed from the initial model. There are several additional classes in this metamodel for specifying different aspects of the system. The Medium implementation specification model located in the top middle area is the actual model class that could be instantiated. The final system source codes are generated according to this model. It carries 2 other models which are Medium design decision specification model and Medium specification model. The design decision model contains all the developer predefined selections for each model transformation. The properties of the final model are all depending on these selections. This model can be viewed as a structured configurations which allows decision modifications that following its structure. Medium specification model is the first model transformed from the abstract definition model that contains all abstract medium information and specified paradigm architecture such as Pub/Sub in our case. It is held for the final source code generation also.

The functions that provided by each component are stated in their own component service interface. The functions in each component service interface are aiming to be generated in actual programming interface as source codes. These functions will be combined into Medium service interfaces as the application service interface (API) for the users of Medium.

Therefore, they can be viewed as internal interfaces and external interfaces respectively.

The Medium class is used to hold shell information of the application without any implementation. It is aimed to provide the system base classes for inheritance. The 3 classes on top of the Medium class are Data Manager, Data Proxy and Data Object which aimed to manage the internal system architectures such as data placement and network resource allocation. They are defined as the non-functional aspects in our definitions.

The classes need to be concerned are Manager class and Role class that located on the left hand side of the figure. Role class is used to define different client roles of the medium, which are Publisher and Subscriber in our case. Manager class is used to define different system operators inside the medium, which are the service brokers and role managers in our case. Therefore, adaptations of the functional variants could be considered as requiring modifications on the proprieties of the manager classes. These models are presented for understanding some basic structures of all the models.

In order to perform component adaptation in an application, refinement process and composition process are required [3, 40]. These processes could be treated as the transformations of different models. Before analyzing each step of the transformations, the packages of all used metamodels should be described.

Metamodel package	**Abstract definition**	**Design decision**	**System specification**
Content	Initial metamodel definitions and all fundamental elements	Decision metamodel definitions and each predefined options	Metamodel definitions in each transformation step and all fundamental elements used in each model

Table 4. Packages of metamodel of Medium for transformation

As shown in table 4, there are 3 metamodel packages in total which are abstract definition, design decision and system specification. The abstract definition package contains the definition of initial model and all the elements needed for the model construction at the beginning. Noticeably, this model is constructed directly according to the metamodel and it is not transformed. System specification package contains all metamodel definitions for each model that transformed in the refinement and composition processes. Of course, these models could be constructed manually also, however, they are transformed and generated automatically by programs to realize the model-driven architecture approach. Thus, the metamodels in this package are used to verify the structure of each transformed model. The decisions of each transformation are stored in a decision model that constructed directly according to its metamodel definition in design decision package. The model could be viewed as an editable configuration of the transformations.

By following the nature of each package, new metamodels of each functional variation point (category) should be added in the system specification package because their actual models

should be transformed. Relatively, the variant options under each category should be added to the decision model. As the new transformations are not affecting the initial model, nothing need to be modified in the abstract definition package.

After the descriptions of each metamodel package, their relations with different transformations could be visualized as figure 4.

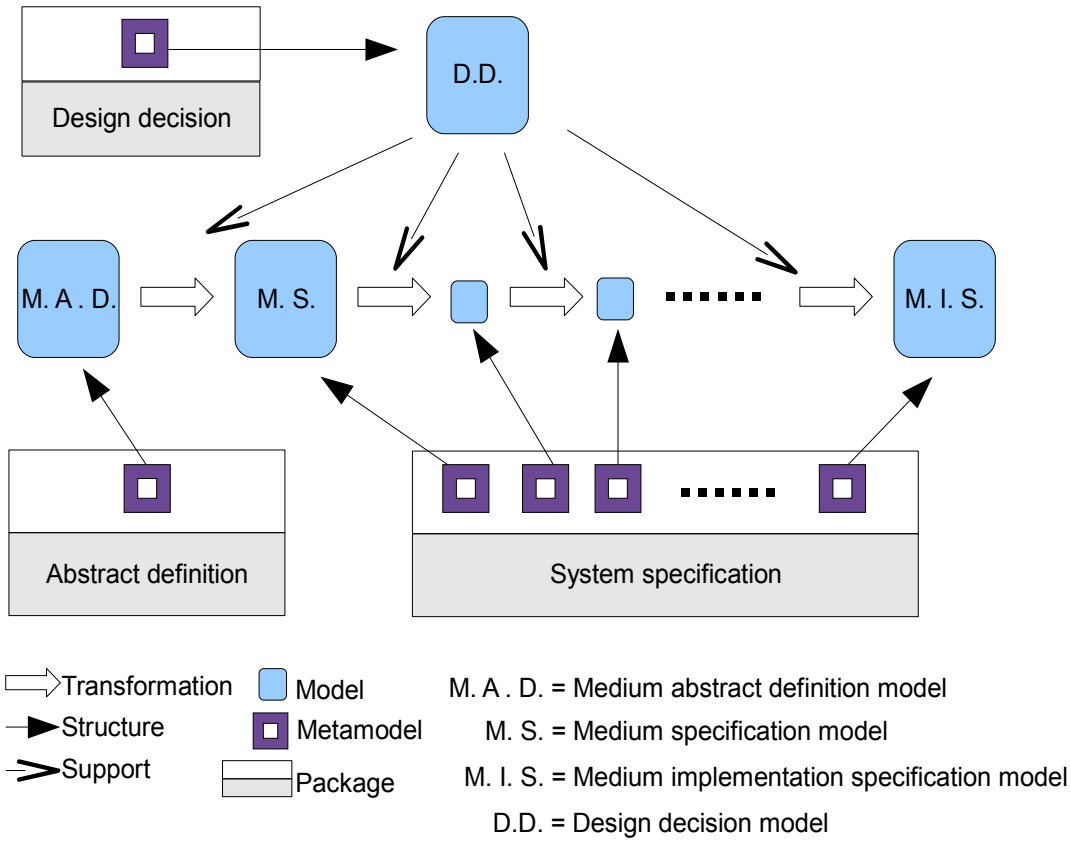

Figure 4. Package references of model transformations

As shown in figure 4, each model (blue square) is structured by the its corresponding metamodel (purple hollow square) in the related package (in rectangle). The abstract definition package only structures the initial model (M.A.D.) where the system specification package structures all the other models including the final completed model. The white arrows are representing the transformations with predefined selections supported by the decision model (D.D.) that defined in design decision package. The name of the metamodels and some actual models are not shown because the figure is aimed to show an abstraction.

M.S. is the first model that transformed from the M.A.D. and it is followed by series of models with their specific transformations. M.I.S is the final model after all transformations and it is aimed for source code generation. The positions of these 3 model are fixed, however, the numbers of model and transformation between M.S. And M.I.S are changeable depending on the needs. All the transformations must be performed sequentially which means that the positions of the models are not swappable except some extra handling are implemented in

each transformation. Nevertheless, this is not suggested to do so because the implementations would be far more complicated according to their closely linked relationship.

The abstraction of our model transformations is explained in this section with detail descriptions of the initial metamodel and the final metamodel. It is not intended to analyze all the other metamodels individually because some of them are out of the scope of this research. They will be briefly introduced in section 3.2.3. The metamodels of our functional variation points are described in next section.

3.2.2 The metamodels of the variation categories and their semantics

According to the definition in 3.1.2, there are 4 functional variation points (categories) identified which are named Event Behaviors (EB), Filter Behaviors (FB), Delivery Behaviors (DB) and Access behaviors (AB). Each category contains the variants identified with the 4 constraints. In order to make these variants as adaptable components in the Pub/Sub medium, 2 types of metamodel have to be defined for each category in the System specification package. They are the Library metamodel (LMM) and Implementation Specification metamodel (ISMM). Library metamodel is aiming to provide a generic structure for the model of each variation point so that different variant implementations could follow a same skeleton. This skeleton provides a structured programming scheme to avoid the system architecture messed up by any unstructured implementation. Implementation specification metamodel is actually the definition of each system model that transformed with particular decision. The models between M.S. and M.I.S. including M.I.S. itself shown in figure 4 are all belong to this nature. Different library models with particular decisions are referenced in each of these ISMMs.

Theoretically, there should be 8 new metamodels in total which 4 of them are LMMs and the other 4 are ISMMs with different structures depending on each category nature. However, 2 questions must be concerned when deciding these metamodels. First, what are the information and restriction should be provided by the LMMs? They should carry all implementation details or just essential definitions? Second, what are the information should be referenced in the ISMMs in order to generate the specified decisions? They should contain entire library models or just their references?

When designing the library metamodels, it should be considered that the purpose of their model instances is to provide skeleton implementation information for each variation points. If all the programming information are stored in these models, the flexibility of any further modifications would be lost. It is because any modifications of the final system source codes are requiring large amount of source code changing in the generator programs. The advantages of software adaptation would be lost [3]. Instead of storing every line of source code information, the variants should be abstracted as objects, which means that modify one variant to another variant within a category could be imagined as objects swapping. The detail implementations could be left to developers to program them manually and flexibly. Therefore, the library metamodel of the each category just have to capable for carrying different signatures of the variants and their generic object-oriented architectures.

By fixing the information architecture of the LMMs in this way, one more aspect should be concerned when designing each variation point LMM. It is the effects of the final system when adapting these functional variations. As mentioned in 3.2.1 with the analysis of figure 3, the adaption of all these functional variations are only affecting the architecture of manager classes which means that they could be modelized in the same structure. The only difference they need is the functions provided in their services interfaces. Figure 5 gives the class diagram of all functional variation library metamodels in a global view.

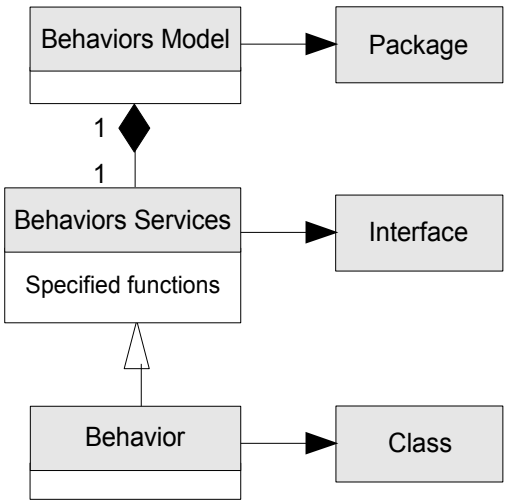

Figure 5. Class diagram of the library metamodel of the functional variation points in global view

As shown in figure 5, each "behaviors" model should carries its own behaviors services with specified functions. Each identified variant (behavior) must implement these services with their own implementations. The "behaviors services" class is an unique interface for each variant class (behavior). By referring these classes to object-oriental architecture, the behaviors model could be viewed as a package where behaviors are individual classes that implement the specified services interface in this package. For example, "Event behaviors" could be the identified model in the package where "Ordered" and "Casual" are the classes that implementing the specified functions in "Event behaviors services" interface.

This global view could be applied to each LMM shown in figure 6 with specified functions signatures in their services interface. The LMM for Event Behaviors is located in top-left corner of the figure, Delivery Behaviors LMM is located in top-right corner where Filter Behaviors LMM and Access Behaviors LMM are located in bottom-left corner and bottom-right corner respectively. These 4 metamodels should be viewed individually but they are shown together just for convenience.

There are 4 functions have to be implemented for each Event Behavior (variant) which related to the management of publications, subscriptions, unsubscriptions and requests. They are defined according to the relationship between the system functionalities and the nature of Event Behaviors. Each Delivery Behavior have to implement 3 functions which related to message delivery controls. Every Filter Behavior have to implement 2 functions for filtering

published journals for suitable subscriptions. Only 1 function need to be implemented for each Access Behavior which concerning authentications and accesses of the system. Considering to maintain a generic information passing interface for all behaviors, “Event” is used as an abstract class for all function parameters. It could be inherited by other particular implementations. This consideration is applied for all functions except some function parameters are not suitable be categorized as event such as a journal. Notably, the return value of all functions are set to void because a callback function must be provided by the corresponding manager for all behaviors. Therefore, these functions need to return nothing to keep their generality. A restriction need to be reminded is that these defined functions in each metamodel are not deletable, however, additional functions could be provided by each behavior (variant) with its own interface that inherit from each global services interface.

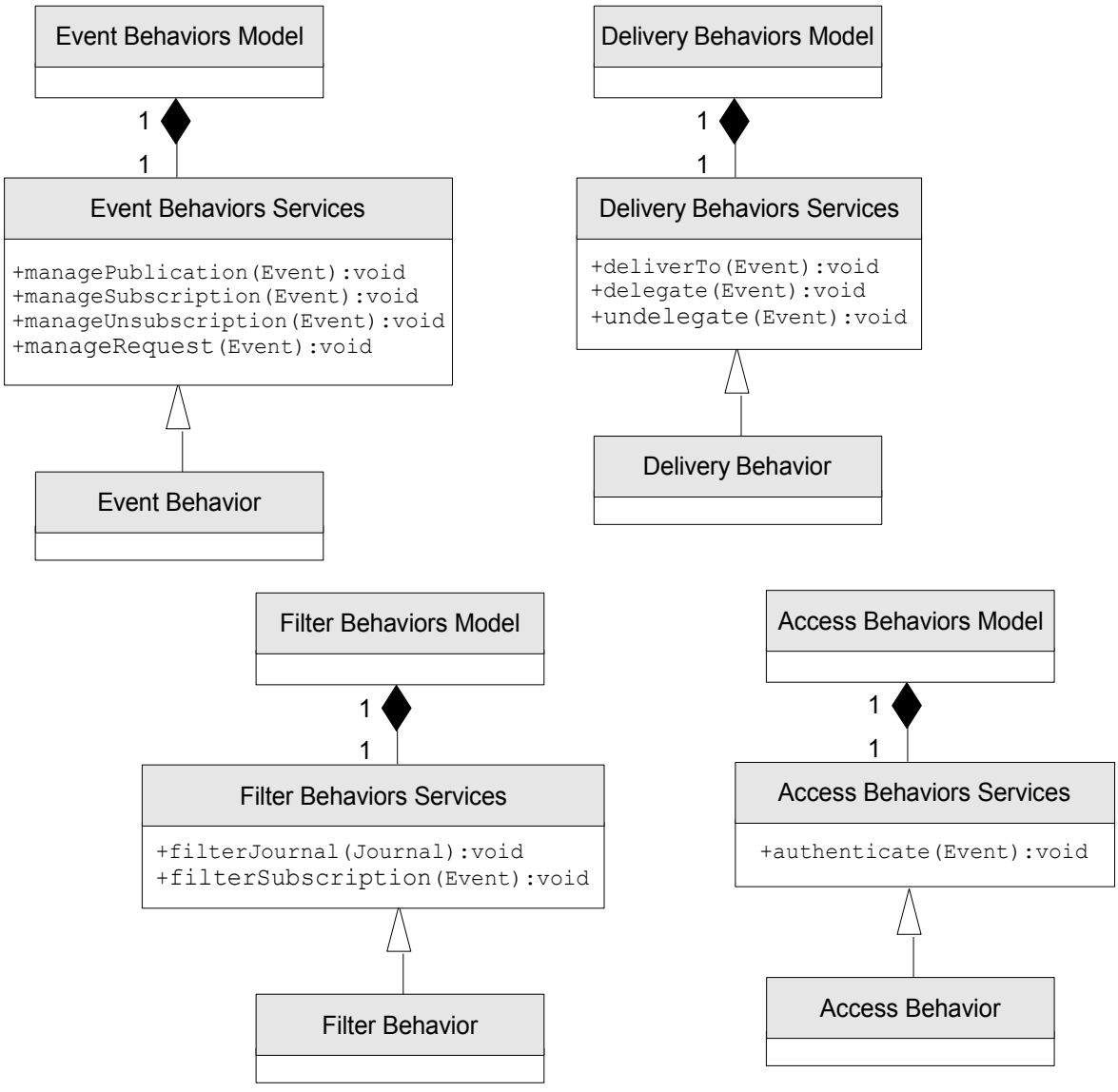

Figure 6. Library metamodel class diagrams for 4 different functional variation points (categories)

When designing the implementation specification metamodels, it should be considered that the purpose of their model instances is aiming to carry the specified decision for each behaviors category, which means that only the references of those selected behaviors are needed to be carried. Also, these models should be allocated in the transformation sequence just before the final model (M.I.S.) shown in figure 3, because the non-functional variants of the Pub/Sub medium should be established before them so that their proprieties are very similar to the M.I.S. According to this characteristics, the simplified ISMM of each functional variation point are defined in figure 7.

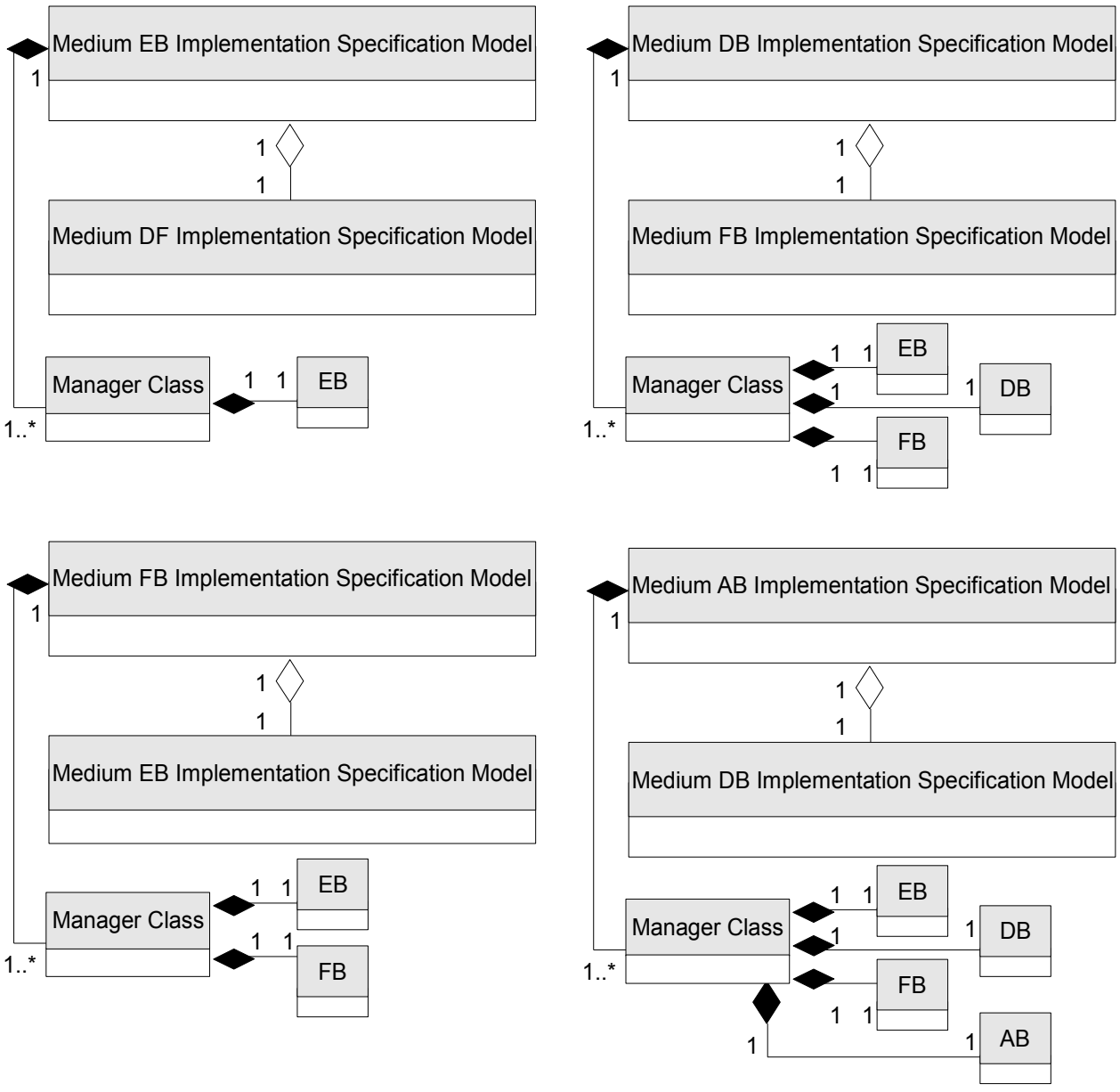

Figure 7. Simplified implementation specification metamodel class diagrams for 4 different functional variation points (categories)

The ISMM for Event Behaviors is located in top-left corner of the figure, Delivery Behaviors ISMM is located in top-right corner where Filter Behaviors ISMM and Access Behaviors ISMM are located in bottom-left corner and bottom-right corner respectively. These 4 metamodels should be viewed individually also.

They are called simplified version ISMM because each metamodel are actually carrying all the same proprieties as the final system model shown in figure 3. However, some of the proprieties are chosen not to be displayed in the figure for visualization convenience. Also, there are nothing changed with them. The Manager Class and a reference model are selected be shown in each metamodel. As displayed in each ISMM, a different model is referenced instead of the Medium specification model. It is indeed the previous model that they are transformed from. The transformations are in ordered sequence as mentioned in figure 3 and the functional variant transformations are ordered as: EB model is the first, FB model is the second, then DB model and then the final is AS model. This order is selected according to the operating structure in corresponding manager classes. The entire transformation sequence will be analyzed in section 3.2.3.

By this transformation order, it could be understood that data format (DF is the non-functional variation identified in [41]) is the previous variation point of Event Behavior so that EB model should hold a DF model. And this is also the reason why the FB model holds a EB model and so on. EB, FB, DB, AB classes shown figure 7 are representing each specified behavior class referenced by the manager class. The references are increased by each transformation according to the order. Thus, the AB model holds all the specified behaviors. It should be known that there are 2 types of manager which are Service Brokers (internal managers) and role managers (which are publish manager and subscriber manager in our case) as mentioned before. The affected managers are only the Service Brokers for our functional variant transformations. Only 1 behavior (variant) implementation can be selected for the manager class under each category. This situation will be clarified in the prototype experiment in section 4.

After defined the LMM and ISMM for each variation category, the last thing have to do is modify the design decision model so that it can carries the functional variant options. Figure 8 shows the modified Design Decision metamodel for the functional variation transformations. The Medium design decision specification model is the actual model that can be instantiated for carrying different configurations. “Service interface implementation choice” contains different variant selections for the global functionalities of Pub/Sub Medium such as Publish, Subscribe and Unsubscribe. The functional variations that we identified are actually related to the “Data Resource Implementation Choice” which provides internal architecture decisions of every resource in the system. As shown in the figure, it holds the additional behavior choice once for each variation category. These classes of choice are defined in the Design decision package. Each choice holds the specified selection such as “Ordered” under “EB Choice” or “Instant” under “DB Choice” depending on any available implementations. Then, they could be used for the related transformations. These selections are predefined manually by developers and changeable for different implementation requirements.

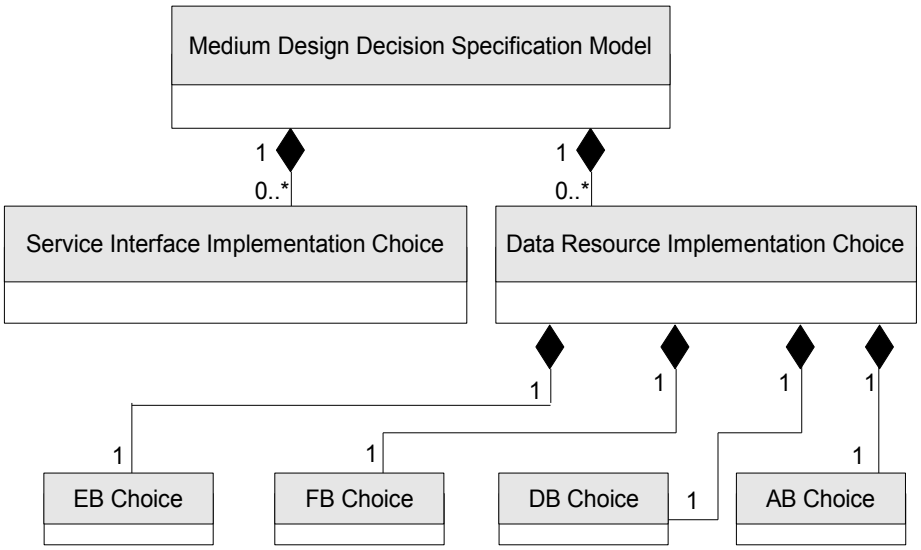

Figure 8. Design decision metamodel class diagram

After established all these metamodels and decisions for the functional variants, the transformations with MDA approach is ready. The details of the entire transformation process (refinement process and composition process in the viewpoint of component adaptation) will be analyzed in next section.

This section is aimed to provide a conceptual architecture of the newly defined elements, therefore metamodels of the variation points are analyzed instead of their actual instances because metamodels are more useful to clarify their structures and restrictions. The actual instances will be shown in the prototype experiment section (section 4) with specified instance proprieties.

3.2.3 Sequence of transformations in the viewpoint of MDA

The details of the entire transformation process will be analyzed in this section with the viewpoint of Model-Driven Architecture. It is important to understand the concept of the process in order to know how it works in the prototype presented in section 4.

Figure 9 presents the details of each transformation and model for constructing the Pub/Sub Medium. It is a detailed version of figure 4. There are 11 models in total and they are structured by their corresponding metamodels (figure 7) defined in those 3 packages that mentioned in figure 4. The order of them is starting from left to right and their full names are indicated by referring to the abbreviations. The abstract type (A.T.), data protocol (D.P.) and data format (D.F.) models are the non-function variation models. The specification and architecture models are aiming to specify the internal architectures for the Pub/Sub Medium. These model are defined in [41] and will not be provided detail analysis here because it is out of scope of our research.

The 4 functional variation models will be added in between the final model (I.S.) and the non-functional variation model (D.F.). In consequence, there are 5 new transformations need to be created which are D.F. to E.B, E.B. to F.B, F.B. to D.B, D.B. to A.B, A.B. to I.S..The original transformation D.F. to I.S. will be deleted. Except the initial model (A.D.) is instantiated manually, all other models are transformed automatically. Each model contains a previous model for referencing related aspects and that model is where it transformed from, except A.D., which has no previous transformation. The model referenced by I.S. is S. instead of A.B. because I.S. has no further transformation and S. is needed to be held for source code generation.

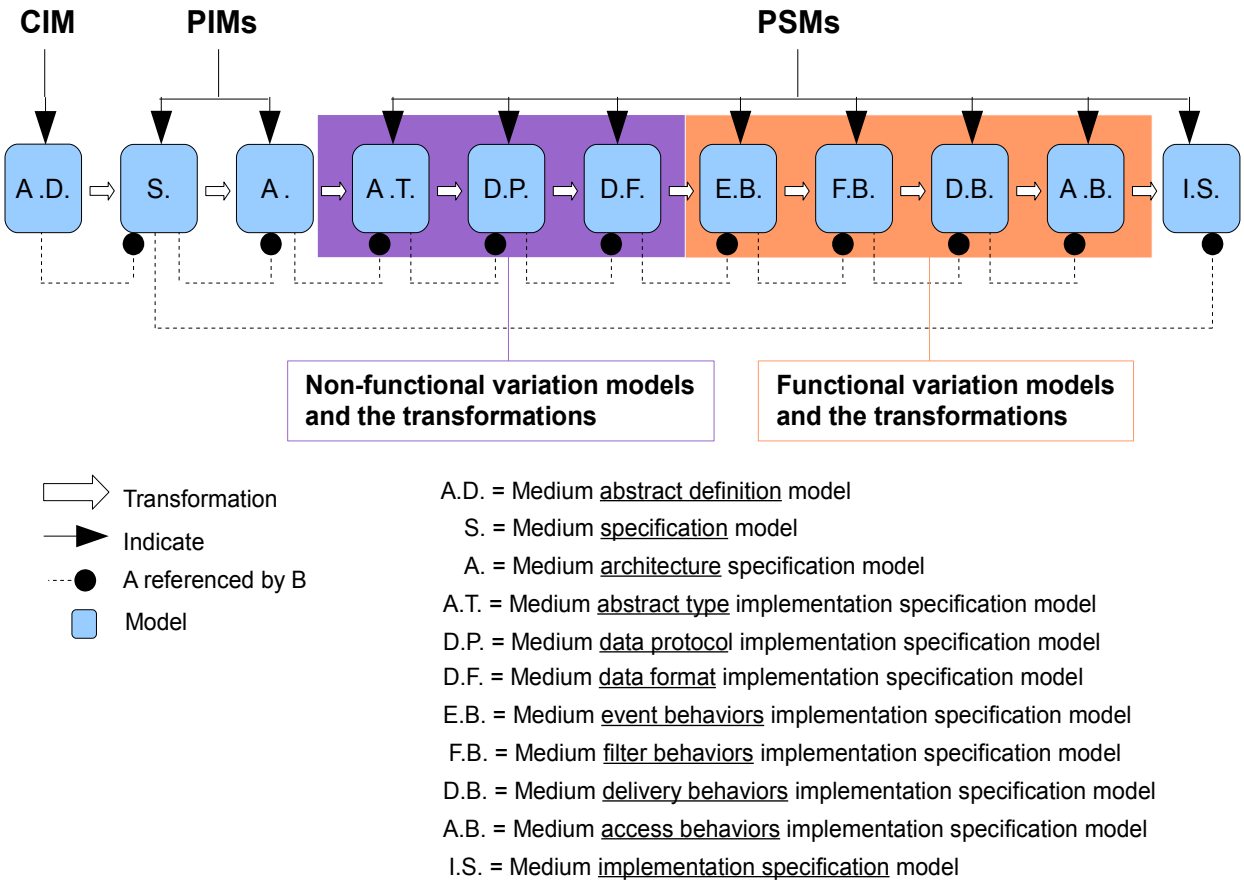

Figure 9. Entire transformation process in MDA approach of current Pub/Sub Medium

In the viewpoint of MDA, 3 different types of models are shown in figure 9 which are computation independent model (CIM), platform independent model (PIM) and platform specific model (PSM). According to the official MDA definitions [24], CIM is a model describing the situation in which the system will be used. It shows the system in the environment which it will operate. It is aimed as an aid to understanding what the system is expected to do. CIM is also required to be traceable by PIM and PSM and vice versa. PIM is aimed to describes the system, however, it does not show details of its used platform. One or multiple particular architectural styles of the system can be applied to it. PSM is produced by transformation and carrying the same proprieties of PIM but also specifies how the system

makes use of the chosen platforms. A PSM may carry more or less detail depending on its purpose.

In figure 9, A.D. is indicated as a CIM that provides the fundamental system information. It describes an abstraction of distributed systems with different roles and resources. S. and A. are indicated as PIMs that provide the the information of Publish/Subscribe paradigm for defining the distributed system in Medium architecture. They analyze the specified proprieties and construct a suitable architecture. Although MDA is specified for platform independent to transform the implementation in different types of programming languages and our transformation process could not be described in exactly the same way, the platforms in our case could be imagined as variation points, and different variants are just similar to different programming languages. Thus, all the models from A.T. to I.S. could be classified as PSMs by this expression. They are carrying the specified variant information that decided by each transformation.

Furthermore, the purpose of the transformations could be visualized explicitly in figure 10.

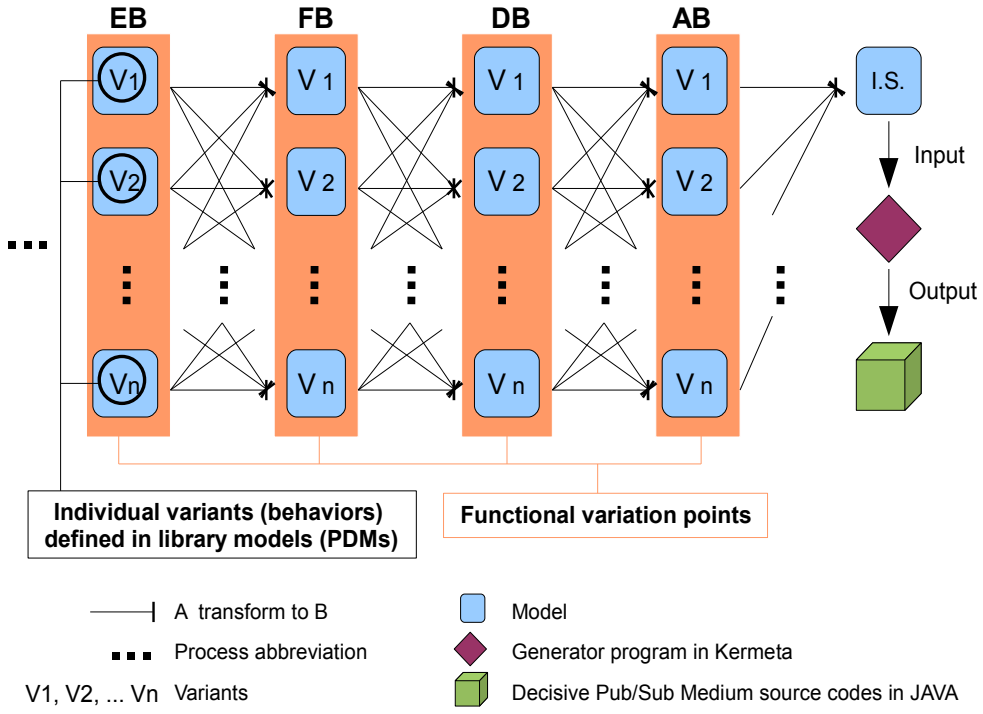

Figure 10. Portion of transformation process in the viewpoint of variant selections

The portion of the functional variant transformations in figure 9 is extracted and extended as shown in figure 10. Transformations from Event Behaviors models (EB) to Filter Behaviors models (FB) and so on are represented by multiple lines which means that the selected variant could be different when each time the models are transformed. The decisions are according to the predefined configurations in Design decision model that mention in figure 8. Each variant in the models of each variation point are reference to individual library models. The structure of these library models are defined by their metamodels that mentioned in figure 6. Each

library model could be classified as platform definition model (PDM) in the viewpoint of MDA. PDM is a model corresponding to a specified technology such as CORBA, .NET, etc. according to the MAD definitions. Thus, these specified technologies could be viewed as the specified variants (behaviors) in our case. Each transformed functional variation model could contains only 1 selected behavior. There is only 1 final model (I.S.) could be transformed after all other transformations because there is no option could be chosen in the final transformation. The I.S. model is treated as an input of the generator program to generate the decisive part of Pub/Sub medium source codes.

Section 3.2 described the entire modeling process of the functional variants that identified and finalized in section 3.1. These functional variants are retrieved according to our research concerns and constraints. The evidences of the identification are supported by the further researches that analyzed in section 2. These contributions are aiming to construct a component adaptive Pub/Sub paradigm with Medium middleware concept. This is main goal of this research. The feasibility of this methodology should also be proved by a prototype that completed in this internship. As the descriptions of the first 2 steps of this internship are completed until the end of this section, next section will describes the final step - the adoption of this methodology to a prototype.

4. Experiments of the prototype

Different aspects of the actual development process are analyzed in this section. All the developments are according to the approach that constructed in section 3.

4.1 Concept of the prototype

In order to realize the solution analyzed in section 3, it must be adopted to an existing prototype base. This base prototype is built for the research of Medium with Pub/Sub architecture and its non-functional variants. The solution must be approached appropriately to suit for the base architecture. Before analyze the adoption approach, there are 2 considerations have to be stated due to the research limitation.

First, the solution mentioned in section 3 is not fully implemented due to the resource and time limit. Only 3 behavior categories will be implemented which are Event Behaviors (EB), Filter Behaviors (FB) and Delivery Behaviors (DB). Access Behaviors category will not be implemented due to the priority importance. Second, only 5 variants in table 3 will be implemented in total due to their result significances, which are “Ordered” and “Casual” under EB, “Topic-based” under FB, “Instant” and “Periodic” under DB. Although there is only 1 option for Filter Behaviors which is topic-based filtering, the solution architecture is constructed for further implementations. The adoption results of these variants will be discussed in section 4.4.

Figure 11 depicts the prototype implementation approach with these considerations. The entire solution is combined by 2 programs which are the Kermeta model transformation program presented on the top-left corner and the JAVA Pub/Sub Medium system presented on the bottom-right corner. The Kermeta program contains 3 component packages as shown. Their presentation from top to bottom is meaning that the metamodels structured the model instances which will be used by the transformation and generation programs. The new items for the functional variations will be added to the corresponding packages as shown on the top-right corner.

The JAVA program contains 3 original packages as shown. Their presentation from top to bottom is meaning that the basic architectures and internal communications are provided by the Kernel package which will be used by the decisive package in the middle and operated by the testing package at the bottom. A brand new package is implemented for the functional variants and it will be put between the Kernel package and decisive package. 2 new graphical user interfaces will be created also for Publishers and Subscribers to show the results of variants adaptations. These interfaces are not aimed to provide full Pub/Sub functionalities at industry level but rather to show the ability of the adaptations. They could be modified to provide more functions for other purposes. Notably, the source generators in the Kermeta program will only generate the source codes in the decisive package. It is because full source code generation is not an efficient solution when modifying one line of system source code require ten lines of generator source codes. Therefore, generated source codes should be aimed for steering the entire target program.

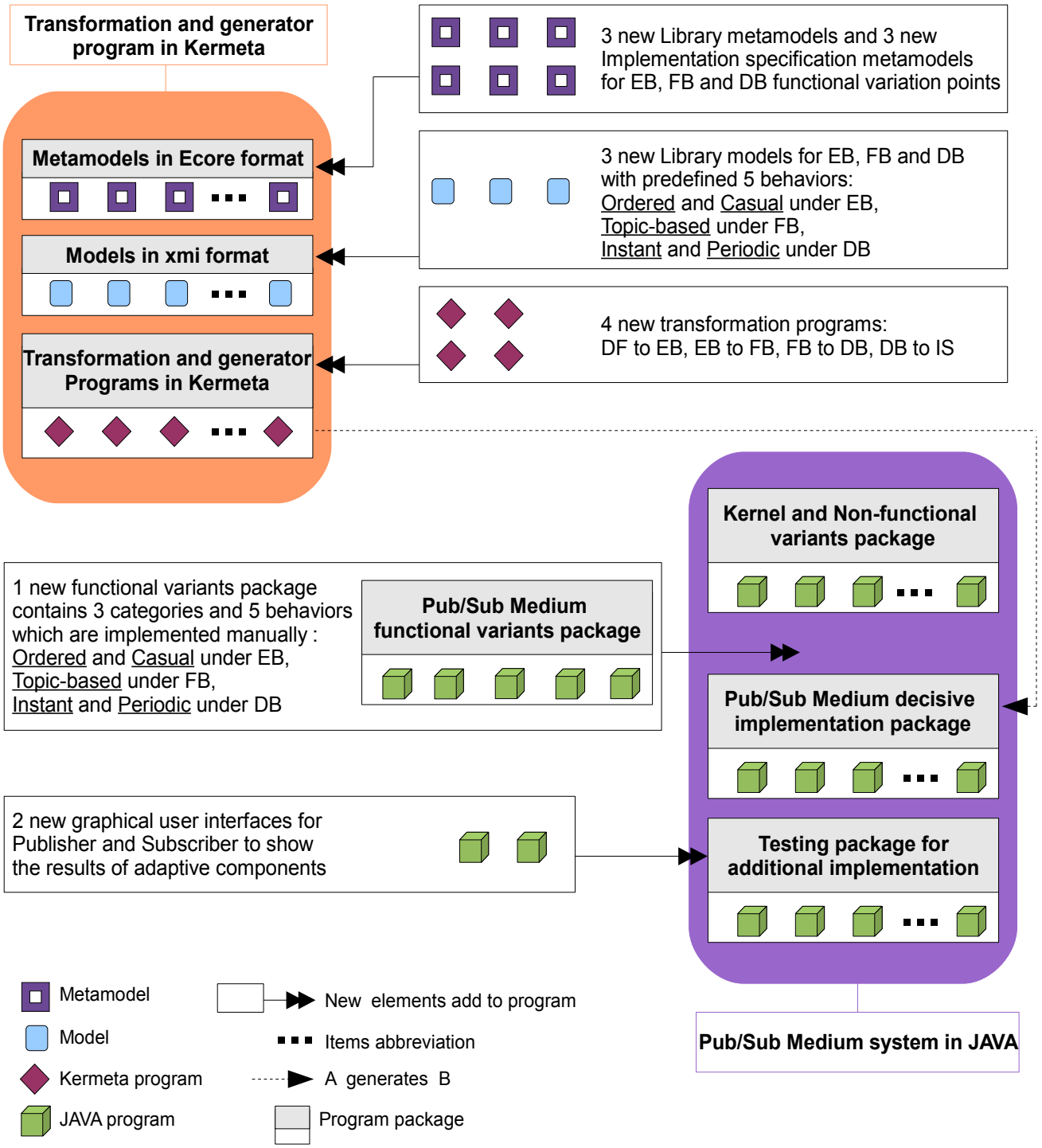

Figure 11. Prototype implementation approach and the programs relationships

4.2 Developments of Kermeta

In order to adopt our solution in to the prototype as shown in figure 11, the Kermeta program have to be modified first.

4.2.1 Metamodels and Models

System specification metamodel ecore file have to be modified shown in appendix A figure ii. 3 library metamodels and 3 implementation specification metamodels are created in this package for EB, FB and DB shown in appendix A figure i. Then, corresponding library model instances need to be created according the structures in the metamodels. Appendix A figure iii shows these models and all initial model instances in xmi format (type of XML format). The contents of the new library models are shown in figure iv. There are 5 variants in total. The contents of the decision model have to be changed also, however, instead of giving its general contents, the model contents are presented with specified scenario settings in appendix C figure i and figure ii. The transformation process is ready to progress when these elements are constructed.

4.2.2 Transformation process

When the settings in 4.2.1 are all ready, the transformation process could be progressed with the help of all transformation programs written in Kermeta that shown in appendix A figure v. Each transformation could be progressed individually or the entire process could be performed at once by using the final model transformation program (create medium implementation specification model.kmt). All transformation outputs are shown on the right hand side in appendix A figure vi, with 9 transformated models in total shown on the left hand side. These models are carrying the specified variants that chosen in the decision model.

4.3 Developments of JAVA

After finish the constructions in the Kermeta program, implementations and modifications have to be done for cooperating with the output of the Kermeta program. These implementations are following the approach that shown in figure 11.

4.3.1 Pub/Sub Medium structure

The packages of the modified JAVA program is shown in appendix B figure ii. There are 8 new packages added under publish subscribe domain in total, where 3 of them are the base packages for each variation category and 5 of them are the defined variants that inherit from the base packages. It should be mentioned that the word “manager” is added to the name of each package for maintaining an unified program style. No additional meaning is added to the variants by this word. 2 packages that generated by the Kermeta program are highlighted which are Pub/Sub specification package and Pub/Sub implementation. They are carrying object shells source codes and the actual implementation source codes respectively.

Appendix B figure i shows the contents of the variants under each each category package. They are all carrying an individual interface that inherit from the global interfaces of each category. The variant objects have to implement the functions that defined in its interface and the global interface depending on each category. This architecture is structured according to the library metamodels that defined in section 3.2.2.

All finalized objects that can be instantiated are shown in appendix B figure iii. 2 new client GUIs are highlighted. There are 4 event objects in total which are PublishEvent, SubscribeEvent, UnsubscribeEvent and NotifyEvent for passing different information between objects. The are all inherit from the base Event class. The classes in this package are aimed to be executed.

After all the developments and testings in the Kermeta program and JAVA program, different scenario of component adaptation with MDA approach could be shown. Nevertheless, 2 important processes should be explained before the scenarios show case.

4.3.2 Important processes in sequence diagram

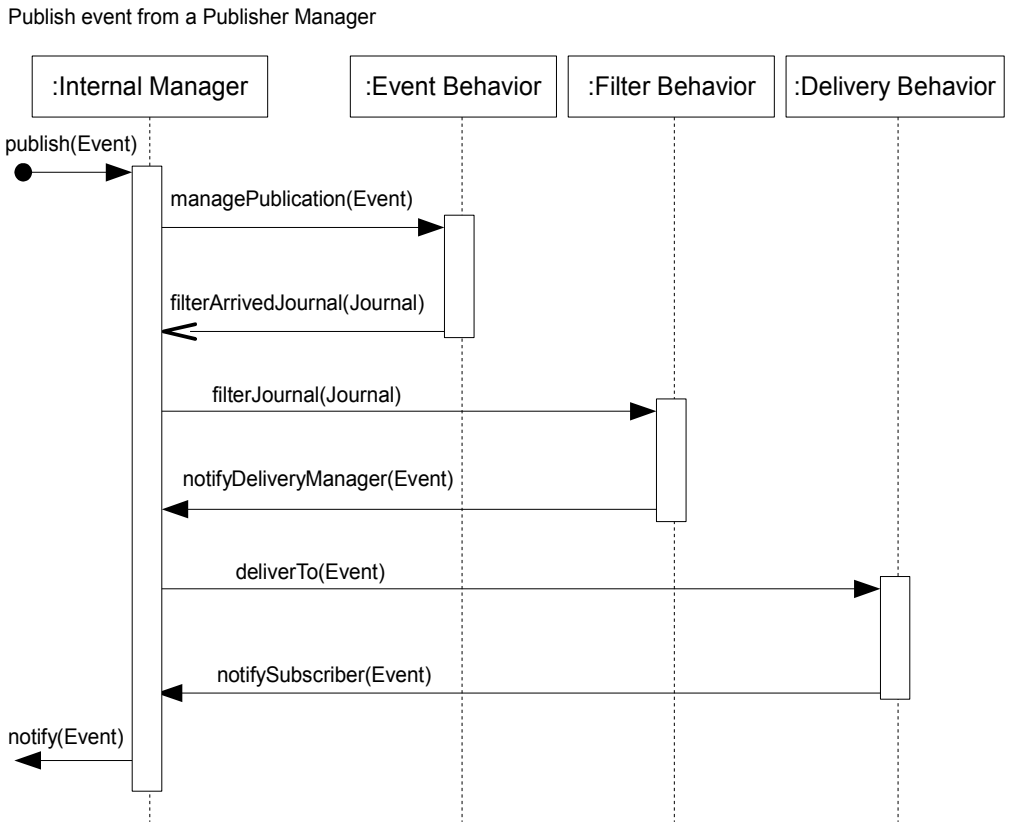

Figure 12. Sequence diagram of a publish event in an internal manager

The important processes “Publish” and “Subscribe” are selected to be analyzed for understand the relevant components interactions. Figure 12 is a UML sequence diagram shows the interactions between an internal manager and our functional variant objects when receiving a publication from a publisher manager. It shows that the communication between each functional variant objects are independent so that the implementations of each variant are flexible. The main controls are depend on the internal manager (Broker). The process is progressed sequentially from EB to DB and this is the reason why the transformation order of the functional variations is chosen like that. The function call filterArrivedJournal(Journal)

before the function call to Filter Behavior is performed asynchronously to maintain the decoupling characteristic of Pub/Sub paradigm. Thus, the entire process could be separated.

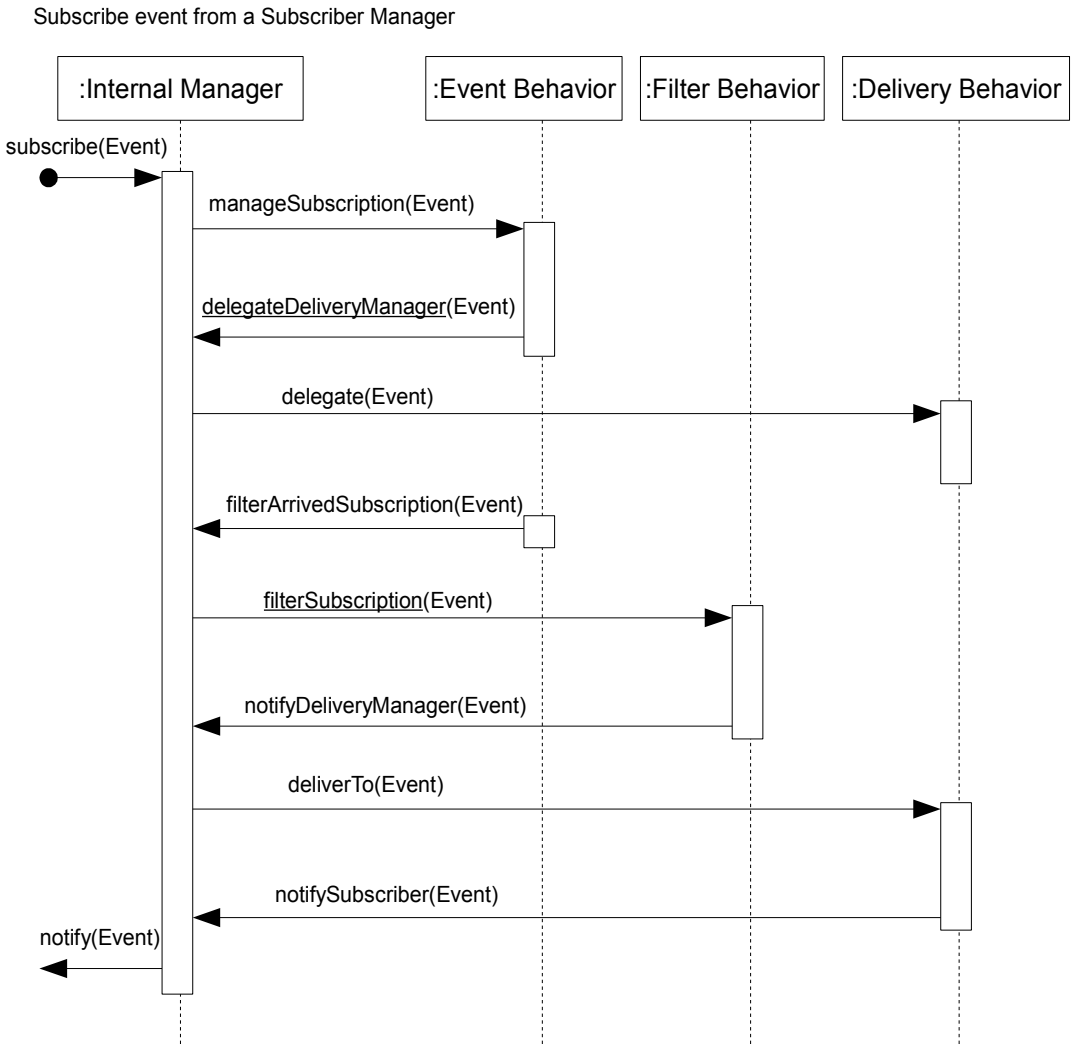

Figure 13. Sequence diagram of a subscribe event in an internal manager

Figure 13 is a sequence diagram shows the interactions between an internal manager and our functional variant objects when receiving a subscription from a subscriber manager. It is similar to the previous process. However, a function delegate(Event) is called before calling the Filter Behavior object because the delivery controls have to be gained first. Another different is the filterArrivedSubcription(Event) is called synchronously to ensure appropriate stored journals are delivered before any new published journals. The parameter Event is not specified to maintain a generic information passing interface. These 2 diagrams are aimed to show the relationship between the functional variants and the internal manager for understand their importances and effects.

4.4 Scenarios of components adaptation

According to the approach stated in section 4.1, there are different combination possibilities of the variant adaptation. 2 of the variant combinations are selected to be analyzed because of there significant differences, which are combination of (Casual under EB, Topicbased under FB, Instant under DB) and combination of (Ordered under EB, Topicbased under FB, Periodic under DB). The meaning of each selected variants are analyzed as follow.

Let "E" be a publication event with "n" number journals, "pt" donates the publication time of a journal from a publisher in an event and "at" donates the arrived time of that journal to a subscriber, "L" donates the loop period of message delivery in periodic. If there are 2 publication events:

$$pt_{a1}, pt_{a2} \ldots pt_{an} \in E_a \quad \text{and} \quad pt_{b1}, pt_{b2} \ldots pt_{bn} \in E_b$$

Variant "Ordered" ensures

$$at_{a1} < at_{a2} < \ldots < at_{an} \quad \text{and} \quad \forall\, at_a < \forall\, at_b \vee \forall\, at_b < \forall\, at_a$$

where variant "Casual" cares nothing. Variant "Ordered" ensures the journals order within a publication event and the atomicity of each publication event. There is no sorting mechanism for all entire events as all publications from different publishers are processed concurrently.

Variant "Instant" ensures

$$\forall\, pt_a \simeq \forall\, at_a, \forall\, pt_b \simeq \forall\, at_b$$

where variant "Periodic" ensures

$$\forall\, at_a \leqslant \forall\, pt_a + L, \forall\, at_b \leqslant \forall\, pt_b + L \quad .$$

Variant "Instant" ensures each journal delivered instantly after it is published. Variant "Periodic" ensures each journal delivered no more than the cycling period time of each subscriber.

In order to shows the scenarios above, the decision model have to be configured for each scenario as shown in appendix C figure i and figure ii. After setup these configurations, the transformation process that analyzed in section 4.2.2 have to be performed once for each scenario. Their final model (MISM) are shown in appendix C figure iv and figure v. Their differences are notified by the highlighted areas (in blue and gray colors). When the final models are ready, their source code generation could be progressed. It should be mentioned that each scenario is progressed separately and they are analyzed together just for convenience. Appendix C figure iii shows the generation output messages and part of the source code differences for both scenarios. It shows that the source codes of the internal manager are holding different object references under both scenario. This is the adaptation results of the model transformation.

Appendix C figure vi and figure vii are the client GUI screen shots for both scenarios. There are 4 client GUIs shown in each figure which 2 of them are Publishers (left hand side) and 2 of them are Subscribers (right hand side). The controls of them are quite simple. Publishers are allowed to publish multiple journals in one publication. Each journal carries its own organization name, journal theme and contents. These journals will be delivered to suitable subscribers according to their matching of organization name and theme. Subscribers are also allowed multiple selections in one subscription. However, the selections could not be modified during the subscripted period unless the Unsubscribe event is called.

Each publisher in each scenario published 50 journals concurrently with the same organization name and theme as shown. Publisher 1 (top left corner) published number 1 to 50 as the content of each journal sequentially according to the number of that journal. Publisher 2 (bottom left corner) published number 51 to 100 as the content of each journal sequentially according to the number of that journal. The results of each subscriber are different in each scenario. In the case of appendix C figure vi, 2 subscribers subscribed to different themes where the theme of subscriber 1 (top right corner) is different from the published journals so that it received nothing. Subscriber 2 (bottom right corner) immediately received all the contents of the journals from both publishers in-orderly and casually. These are the effects of variants “Casual” and “Instant”. In the case of appendix C figure vii, 2 subscribers subscribed to the same theme and same organization of all published journals. However, subscriber 1 received nothing when subscriber 2 received all the contents of the journals from both publishers orderly and atomically. It is because subscriber 1 made the subscription later than subscriber 2 so that its delivery time is not reached yet. The cycling period could be changed depending on different client needs. These are the effects of variants “Ordered” and “Periodic”. The combinations “Ordered & Instant” and “Casual & Periodic” are also work but they are not selected to be shown in this report due to the content limit.

These 2 scenarios showed the component adaptations of Pub/Sub Medium with MDA approach are successful. It also demonstrates the methodology that was identified in section 3.

The prototype experiments are analyzed in this section. It is the final step of the internship and stated a significant evidence for the researches and methodology that defined in previous 2 steps. Although there are lots more details that could be shown from the prototype, they are not mentioned due to the content limit and their less important representation. Some prospectives for further researches on this topic are recommended in next section before the conclusion of the entire internship.

5. Prospective of further research developments

Some prospectives of the research and development are recommended in this section. They could be considered as further research topics in this area.

5.1 Possible enhancements of the source code generation

Other than full implementation of the solution stated in section 3 (including AB category and all mentioned variants in table 3), source code generation is another aspect that could be optimization in this research. The communications between objects in the JAVA program are required object references. For example, the functional variant objects are required to hold an internal manager reference for function callback. Indeed, their architectures allow different callback approaches such as event raising. It would be more flexible and generic if the source code generation could support different implementation styles. All aspects of the source code generation could be optimized as same as this example

5.2 Model transformation optimization

The current model transformations and source code generations are very complex and they require very large amount of meta-programming. In certain extents, it is due to the limitation of the used language such as Kermeta. Other than replacing it with other modeling tools, some optimization could be developed. For example, a generic custom utility package could be developed for supporting all types of model transformation. This would be very helpful for substantial developments on modern-driven architecture researches. When modeling tools are mature enough, model-driven programming could possible be used popularly.

5.3 The concerns of other medium applications

As a completed methodology of variants transformation is presented and demonstrated in this research internship, it could be believed that this methodology is possible to be applied to some other applications with medium context. For instance, some applications that are intended to provide middleware services, such as a web service server, could use this methodology to retrieve and apply functional variants adaptations in order to simplify their development processes and any continuous developments. The approach that was defined in this internship could be viewed as a generic solution for this type of application.

6. Conclusion

Traditional Pub/Sub paradigm in distributed systems provides scalable benefit for large scale messaging systems. There are different implemented variations provide various functionalities for the paradigm. However, these functionalities are based on separated researches and not combinable. In order to create an adaptive Pub/Sub application, component adaptation should be applied to the paradigm. This internship successfully introduced a solution methodology for achieving the target. Fundamental researches provided the significant evidences for the functional variants identification. By the model transformation characteristic of Model-Driven Architecture, the component adaptations could be applied and performed easily.

The purposes of this research internship are to clarify variants of Pub/Sub system, apply them into relevant models and develop a prototype to demonstrate component adaptations are possible be constructed by using model-driven architecture. There are several significances of this research internship. First, the functional variation points and their variants of current Pub/Sub systems are successfully identified. The importance of those identified categories is not only adaptable with existing variants, but also adaptable for different variants that could possibly be developed in the future. Second, it points out model-driven programming could possible be a popular development method in software engineering industry because of its flexibility and efficiency of software modification. Third, it shown the power of component adaptation could benefit different aspects of software developments, such as developing a distributed system in our case. The results and analysis in this report proved these conclusions significantly.

7. References:

[1] P. TH. Eugster, P. A. Felber, R. Guerraoui, A. M. Kermarrec. *"The Many Faces of Publish/Subscribe"*. ACM Computing Surveys, Vol.35, No.2, pp. 114-131, June 2003

[2] P. TH. Eugster, R. Guerraoui, C. H. Damm. *"On Objects and Events"*. ACM, 2001

[3] J. M. Gilliot, P. K. An, A. Beugnard, M. T. Segarra. *"L'ingénierie dirigée par les modèles pour la conception d'applications à architectures réparties adaptables"*. L' objet, Vol. 8, No. 1, pp. 1 – 26, 2009

[4] A. Ketfi, N. Belkhatir, P. Y. Cunin. *"Adaptation Dynamique Concepts et Expérimentations"*. ICSSEA, 2002

[5] R. Baldoni, R. Beraldi, S. T. Piergiovanni, A. Virgillito. *"On the modeling of publish/subscribe communication systems"*. Concurrency and computation: Practice and Experience, Vol.17, pp. 1471-1495, 2005

[6] R. Baldoni, M. Contenti, A. Virgillito. *"The Evolution of Publish/Subscribe Communication Systems"*. Future Directions of Distributed Computing. Springer Verlag, LNCS 2584, 2003.

[7] G. Mühl. *"Generic Constraints for Content-based Publish/Subscribe"*. Springer Verlag, LNCS 2172, pp. 211 – 225, 2001.

[8] T. Sivaharan, G. Blair, G. Coulson. *"GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing"*. Springer Verlag, LNCS 3760, pp. 732 – 749, 2005.

[9] L. Fiege, F. C. Gartner, O. Kasten, A. Zeidler. *"Supporting Mobility in Content-Based Publish/Subscribe Middleware"*. IFIP International Federation for Information Processing, LNCS 2672, pp. 103-122, 2003.

[10] J. Wang, B. Jin, J. Li. *"An Ontology-Based Publish/Subscribe System"*. IFIP International Federation for Information Processing, LNCS 3231, pp. 232 – 253, 2004.

[11] N. Astley, J. Auerbach, S. Bhola, G. Buttner, M. Kaplan, K. Miller, R. Saccone, R. Strom, D. C. Sturman, M. J. Ward, Y. Zhao. *"Achieving Scalability and Throughput in a Publish/Subscribe System"*. IBM Research Report, February 2005.

[12] M. Caporuscio, A. Carzaniga, A. L. Wolf,. *"Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications"*. IEEE Transactions on software engineering, Vol. 29, No. 12, December 2003

[13] G. Mühl. *"Large-Scale Content-Based Publish/Subscribe Systems"*. Vom Fachbereich Informatik der Technischen Universit□at Darmstadt, 2002.

[14] R, Baldoni, L. Querzoni, A. Virgillit. *"Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey"*. University of Rome, 2005.

[15] N. Bencomo, G. Blair. *"Using Architecture Models to Support the Generation"and Operation of Component-Based Adaptive Systems"*. Self-Adaptive System, Springer Verlag, LNCS 5525, pp. 183 – 200, 2009.

[16] D. Gracanin, S. A. Bohner, M. Hinchey. *"Towards a Model-Driven Architecture for Autonomic Systems"*. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004.

[17] B. Morin, O. Barais, G. Nain, J. Jezequel. *"Taming Dynamically Adaptive Systems Using Models and Aspects"*. IEEE, May 2009.

[18] P. Oreizy, N. Medvidovic, R. N. Taylor. *"Runtime Software Adaptation: Framework, Approaches, and Styles". ACM*, May 2008.

[19] P. C. David, T. Ledoux. *"An Infrastructure for Adaptable Middleware"*. Springer Verlag, LNCS 2519, pp. 773– 790, 2002.

[20] T. E. Bihari, K. Schwan. *"Dynamic Adaptation of Real-Time Software"*. ACM Transactions on Computer Systems, Vol. 9, No. 2, Pages 143-174, May 1991.

[21] E. Cariou, A. Beugnard, J. M. Jezequel. *"An Architecture and a Process for Implementing Distributed Collaborations"*. IEEE sixth international enterprise distributed object computing conference, 2002.

[22] *J. D. Poole*. *"Model-Driven Architecture: Vision, Standards And Emerging Technologies"*. *Workshop on Metamodeling and Adaptive Object Models*, April 2001.

[23] T. Kühne. *"Matters of (meta-) modeling"*. *Softw Syst Model*, pp. 369 – 385, 2006.

[24] J. Miller, J. Mukerji. *"MDA Guide Version 1.0.1"*. *OMG*, June 2003.

[25] E. Bruneton, T. Coupaye, J. B. Stefam. *"The Fractal Component Model"*. The ObjectWeb Consortium, 2002 - 2003.

[26] F. Chauvel, Z. Drey, F. Fleurey. *"Kermeta Language Overview: The Triskell Metamodeling Language"*. *Triskell team IRISA*, January 2007.

[27] M. Milenkovic, S. H. Robinson, R. C. Knauerhase, D. Barkai, S. Gar, V. Tewari, T. A. Anderso, M. Bowman. *"Toward Internet Distributed Computing"*, IEEE Computer Society, 2003

Additional references:

[28] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward. "Gryphon: An Information Flow Based Approach to Message Brokering", IBM T.J. Watson Research Center, 1998

[29] S. Bhola, R. Strom, S. Bagchi, Y. Zhao and J. Auerbach. *"Exactly-once Delivery in a Content-based Publish-Subscribe System"*, IEEE Computer Society, 2002

[30] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon and K. Moody. *"RoleBased Access Control for Publish/Subscribe* Middleware Architectures*"*, ACM, 2003

[31] P. R. Pietzuch and J. M. Bacon. *"Hermes: A Distributed Event-Based Middleware Architecture"*, IEEE Computer Society, 2002

[32] F. Fabret, F. LLirbat, J. Pereira and D. Shasha. *"Publish/Subscribe on the Web at Extreme Speed"*, 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc, 2000

[33] J. Pereira, F. Fabret, F. LLirbat and D. Shasha. *"Efficient matching for web-based publish/subscribe systems"*, 7th International Conference on Cooperative Information Systems, Springer-Verlag, 2000

[34] M. Caporuscio, A. D. Marco, P. Inverardi.*"Runtime Performance Management of the Siena Publish/Subscribe Middleware"*, ACM, 2005

[35] M. Caporuscio, A. Carzaniga and A. L. Wolf. *"Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications"*, IEEE Computer Society, 2003

[36] A. Rowstron, A. M. Kermarrec, M. Castro and P. Druschel. *"SCRIBE: The design of a large -scale event notification infrastructure"*, Microsoft Research, 2001

[37] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowstron. *"SCRIBE: A large-scale and decentralized application-level multicast infrastructure"*, IEEE Computer Society, 2002

[38] M. Glinz. *"On Non-Functional Requirements"*, IEEE Computer Society, 2007

[39] L. Xu, H. Ziv, D. Richardson and Z. Liu. *"Towards Modeling Non-Functional Requirements in Software Architecture"*, Aspect-Oriented Requirements Engineering and Architecture Design , 2005

[40] M.Segarra, J. Gilliot and A. Lee. "*Adaptive variants for Publish/Subscribe systems*", Master research study of University of Rennes 1, 2010

[41] E. C. Kabore. "*Contribution à l'automatisation d'un processus de construction d'abstractions de communication par transformations successives de modèles*", Doctoral research of Telecom Bretagne, 2008

Appendix A: Kermeta development environment screen shots

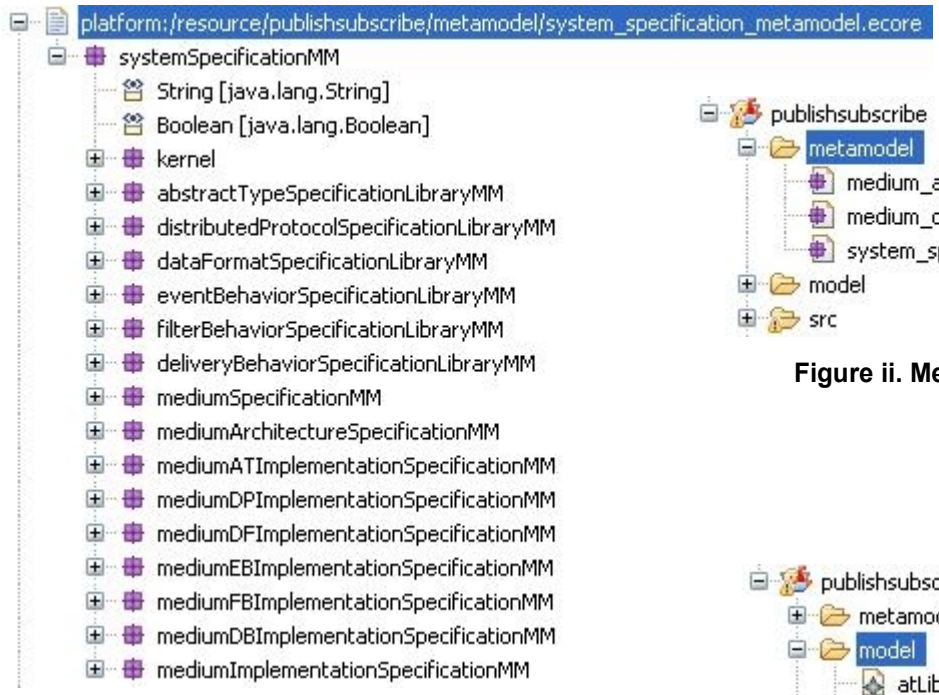

Figure i. Metamodel packages

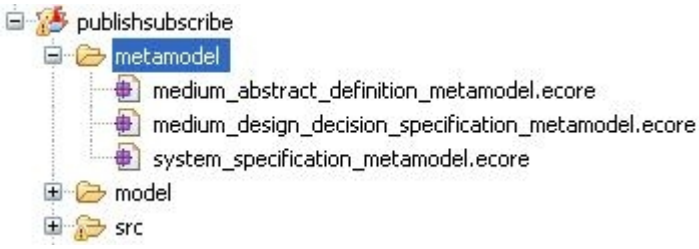

Figure ii. Metamodel packages in ecore files

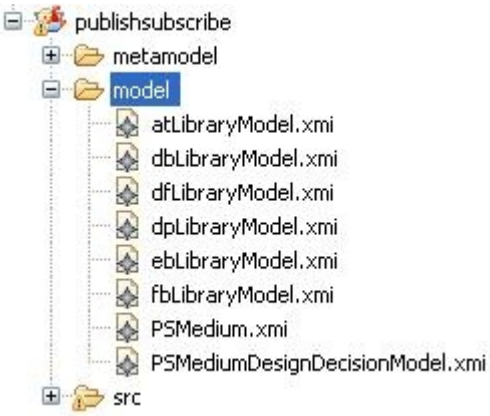

Figure iii. Initital models in xmi format

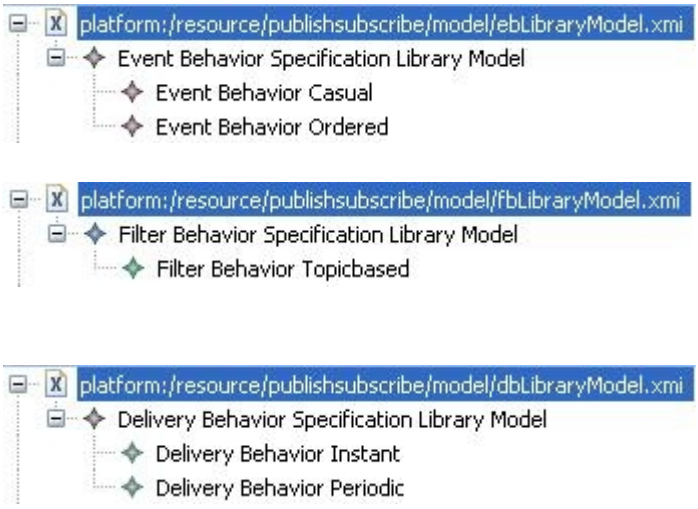

Figure iv. Contents of EB, FB & DB library models

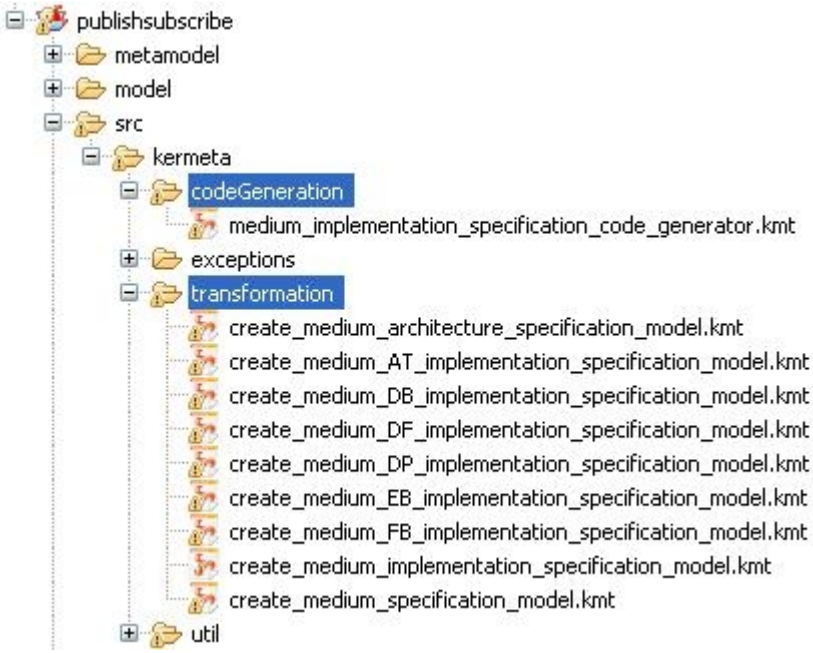

Figure v. Transformation programs and generator program in Kermeta

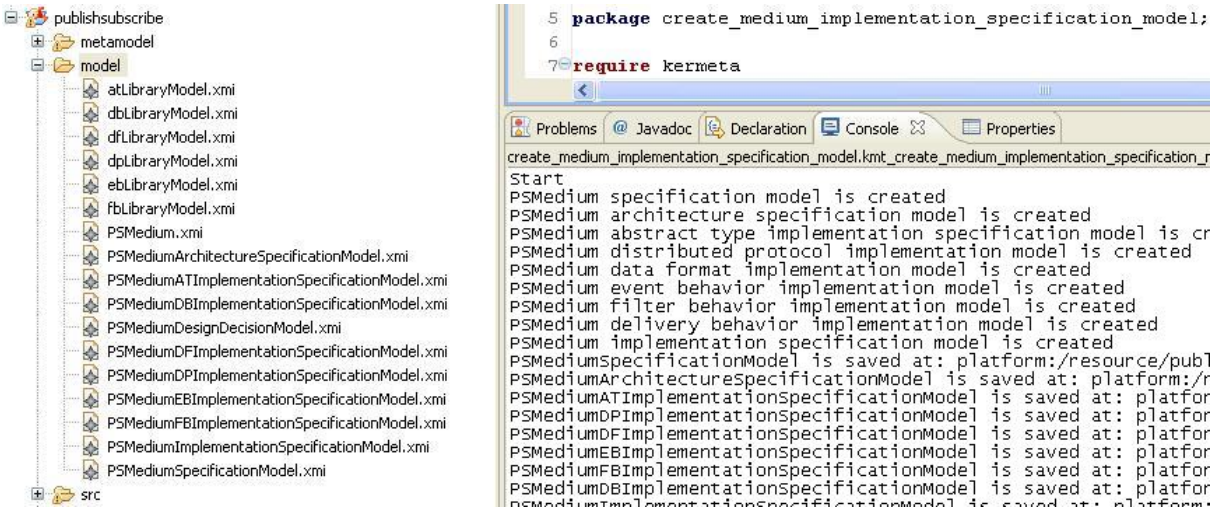

Figure vi. Transformed models in xmi format and result output

Appendix B: JAVA development environment screen shots

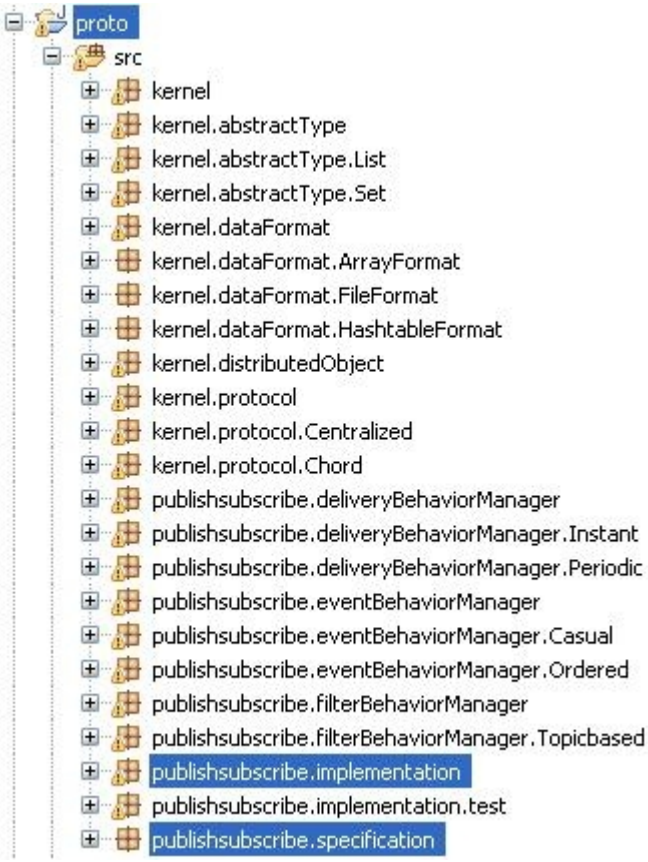

Figure ii. All packages in Pub/Sub Medium with generated packages highlighted

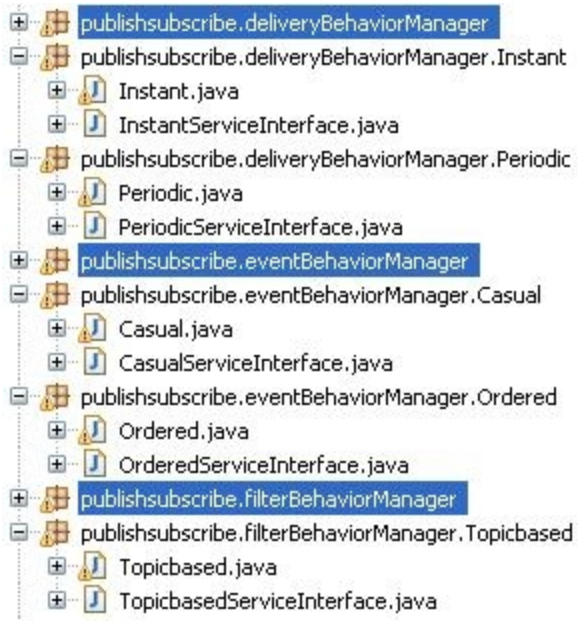

Figure i. 5 variants under 3 variation categories

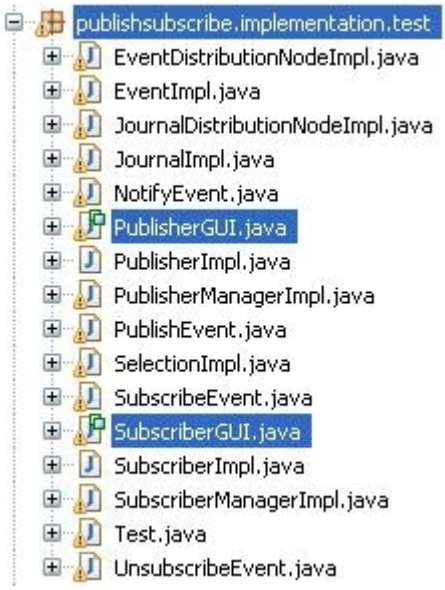

Figure iii. Objects for instantiated in testing pack with new GUI highlighted

Appendix C: Scenarios screen shots

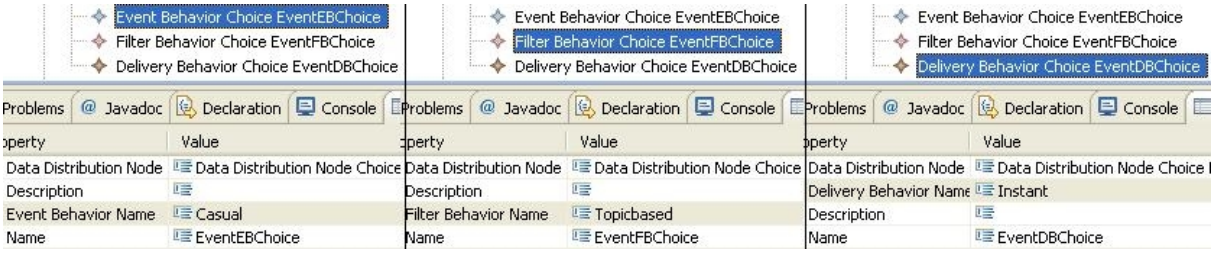

Figure i. Decision model setups for scenario (Casual, Topicbased, Instant)

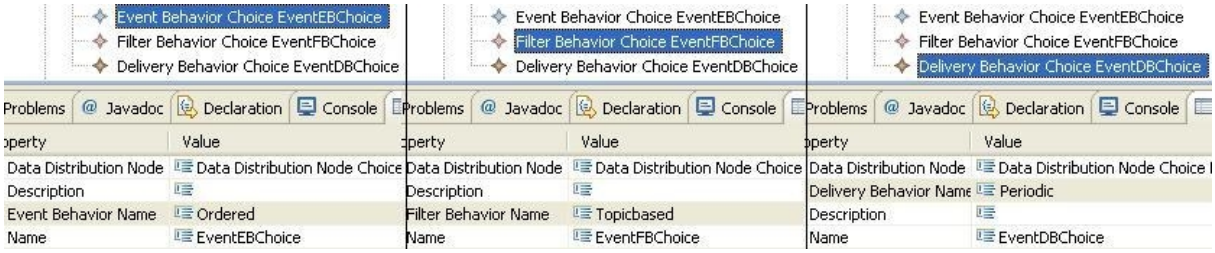

Figure ii. Decision model setups for scenario (Ordered, Topicbased, Periodic)

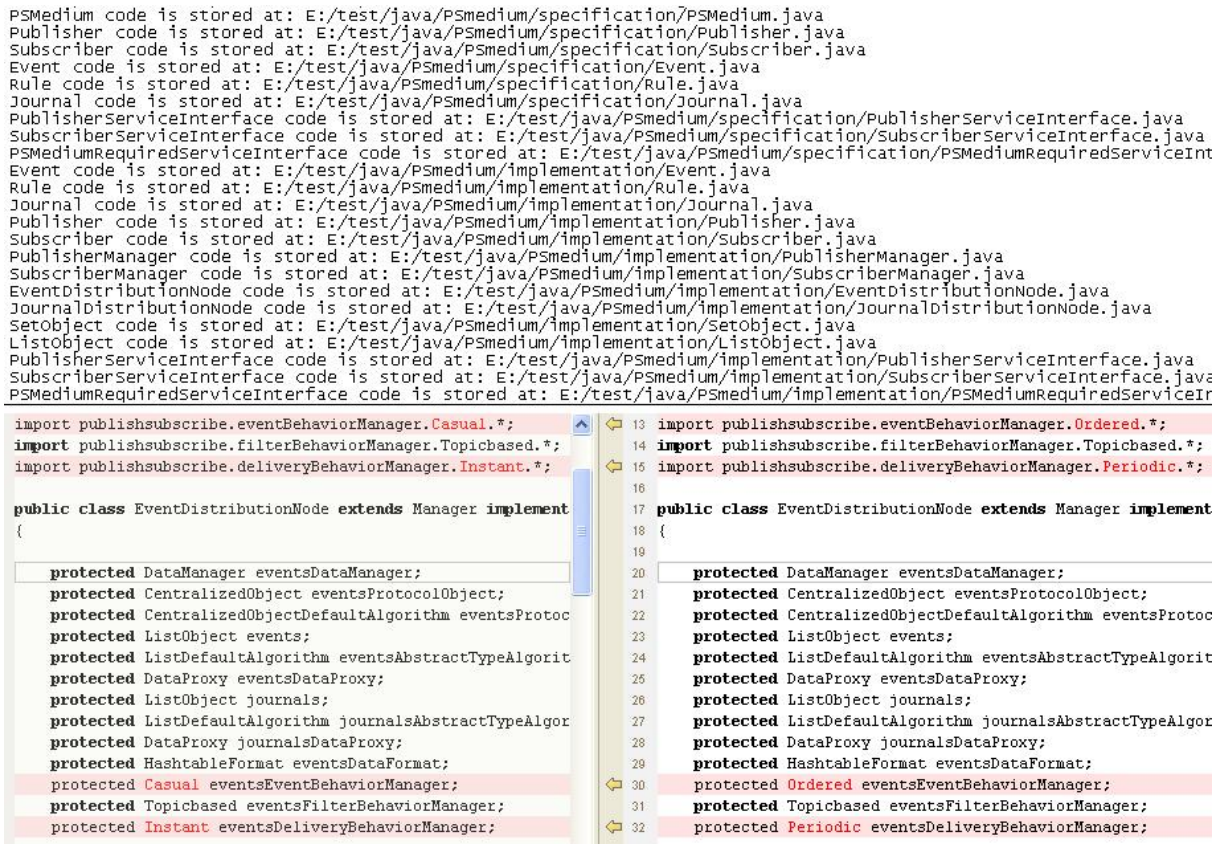

Figure iii. Output messages of the source code generation and their differences

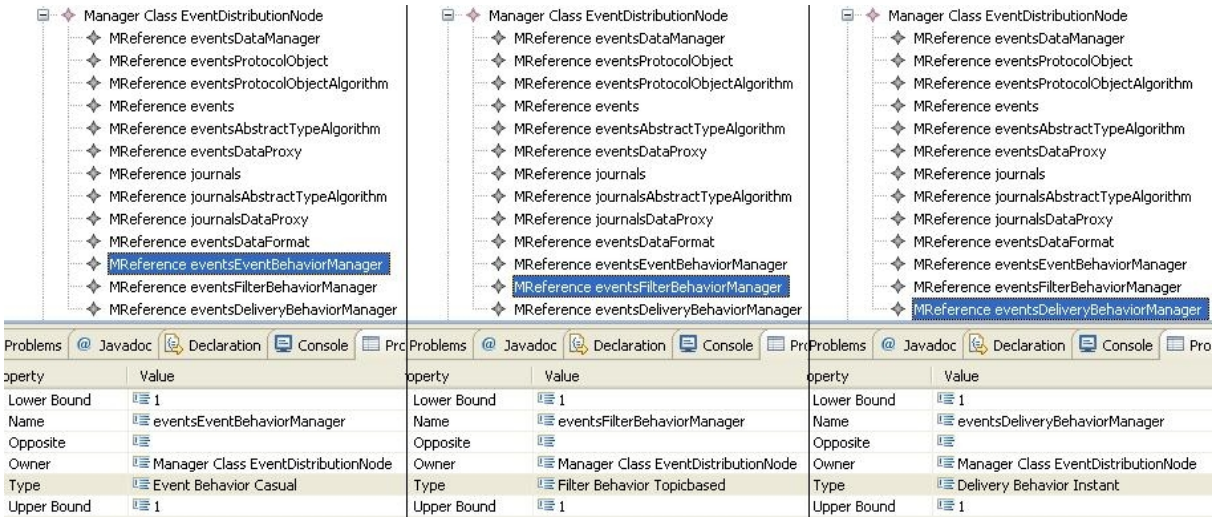

Figure iv. Final model with generated references for scenario (Casual, Topicbased, Instant)

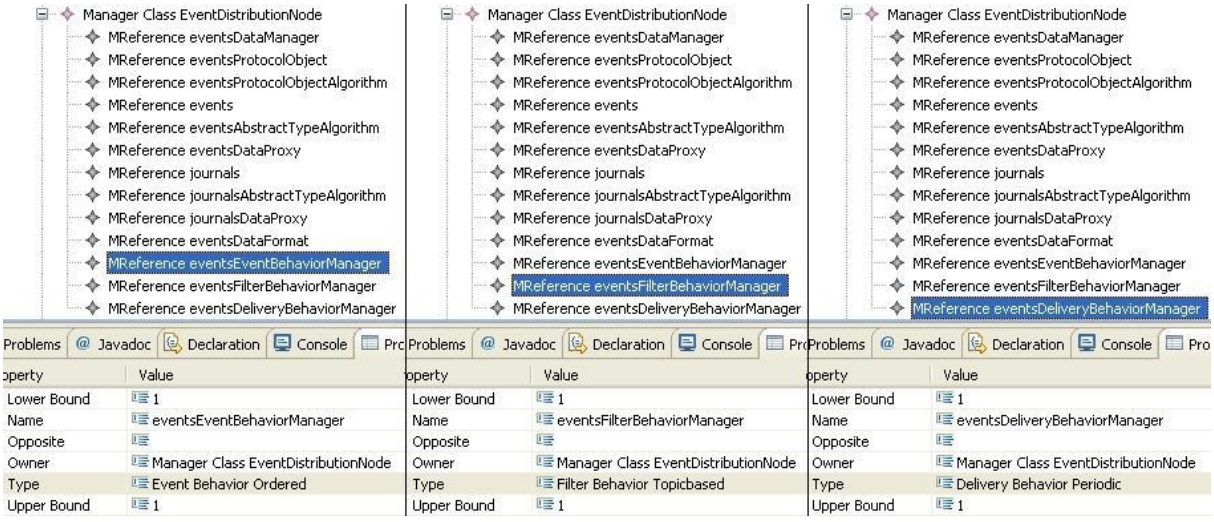

Figure v. Final model with generated references for scenario (Ordered, Topicbased, Periodic)

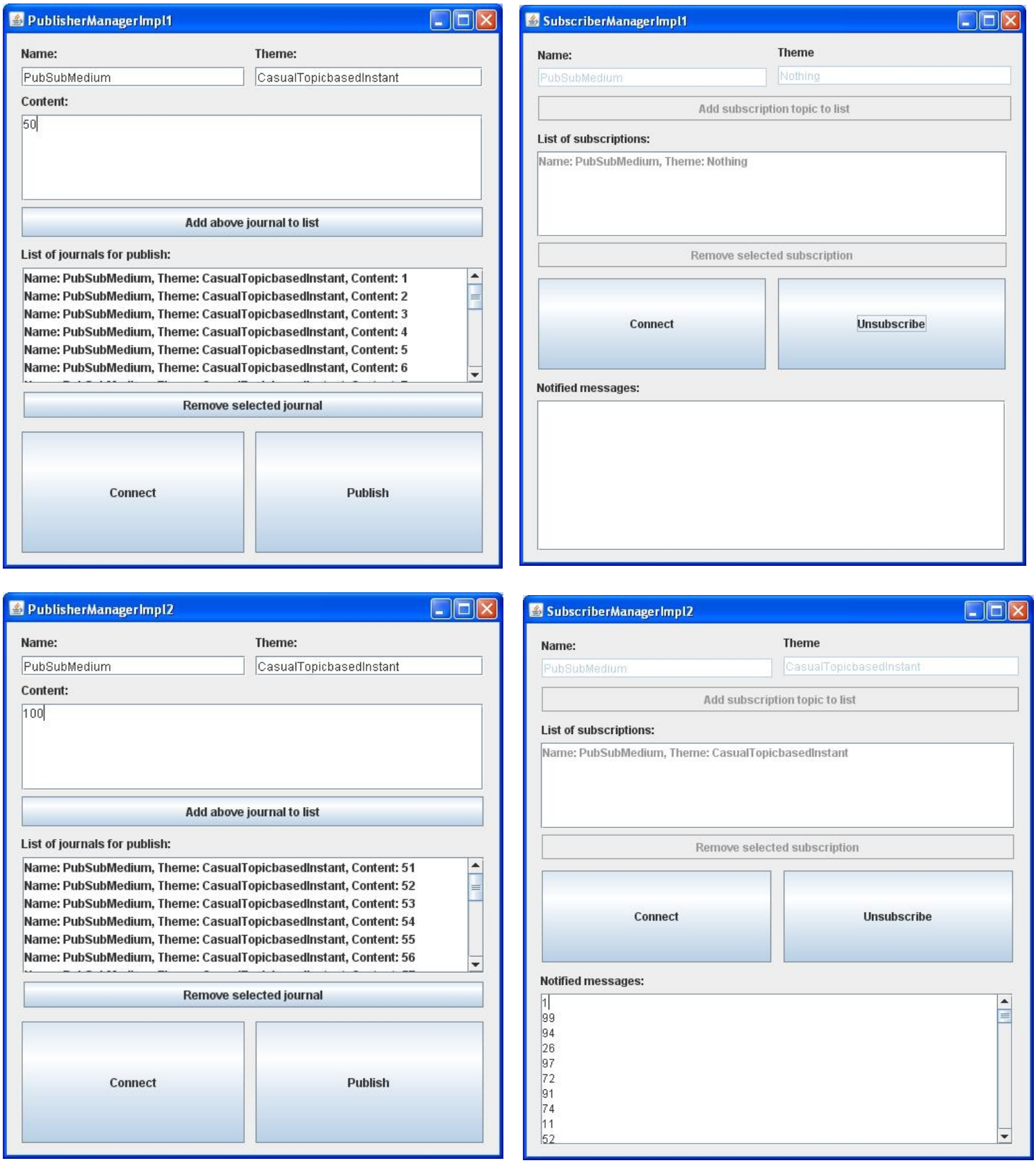

Figure vi. Results of Publishers and Subscribers GUI of scenario (Casual, Topicbased, Instant)

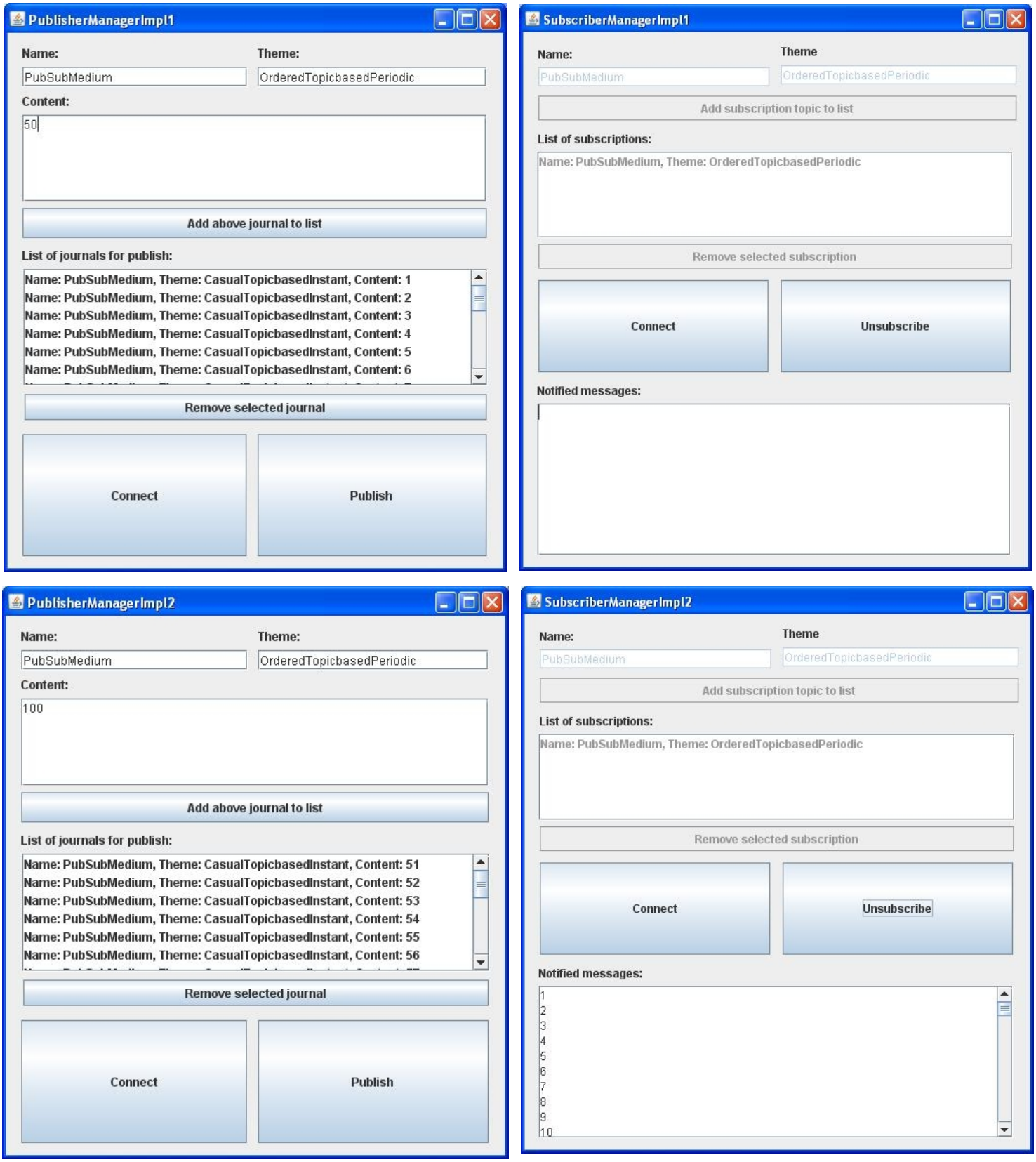

Figure vii. Results of Publishers and Subscribers GUI of scenario (Ordered, Topicbased, Periodic)