HAL
open science

Implementing a Data Distribution Variant with a Metamodel, Some Models and a Transformation

Eveline Kabore, Antoine Beugnard

To cite this version:

Eveline Kabore, Antoine Beugnard. Implementing a Data Distribution Variant with a Metamodel, Some Models and a Transformation. DAIS 2008 : Distributed Applications and Interoperable Systems, 8th IFIP WG 6.1 International Conference, Jun 2008, Oslo, Norway. pp.224 - 237. hal-02161892

HAL Id: hal-02161892

https://hal.science/hal-02161892v1

Submitted on 21 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Implementing data distribution variant with a metamodel, some models and a transformation

Eveline Kaboré and Antoine Beugnard

Department of Computer Sciences, TELECOM Bretagne, Technopôle Brest-Iroise
CS 83818– 29238 Brest Cedex 3, France
{eveline.kabore, antoine.beugnard}@enst-bretagne.fr

Abstract. In this paper, we show how models transformations can be used to implement data distribution features in the software design process of a component. This approach is based on a single metamodel that defines data distribution abstractions and on the design of alternatives that are used to implement each data distribution variants. A model transformation is associated to the metamodel and the component metamodel we consider as the target. We show that this approach facilitates the derivation of different implementation strategies from the model of a component. We illustrate our approach through the example of a distributed communication component software that implements a centralized and two peer-to-peer variants and demonstrates the reusability of the transformation.

1 Introduction and motivation

Models are widely used in sciences and have become an unavoidable tool for software designers and programmers. Models were used in many development methods such as SADT, JSD, etc. They allow the description of different aspects of a system: structural, functional, behavioral, temporal, etc. Models also permit the description of the system to be developed at different stages with various levels of details.

The Unied Modelling Language (UML) is the last avatar of a standard modelling notation. The way models are produced and elaborated is mainly beyond the scope of modelling; it is mainly good-practices, know-how and methods more or less formalized. One of the last great advances in software engineering was the introduction of patterns (especially design patterns) as a semi-formalization of good (or bad) practices.

The formalization and the clarication of the process of elaborating models are the next challenges. Considering the processes of elaborating and rening models as an activity that can be described with a dedicated language is in our point of view, a revolution.

We show in this article how models, metamodels (that can be defined as model types) and models transformations can be used to automatize the design and implementation process of a distributed software component.

We are working on a specific component model dedicated to communication [1,2] and the way to derive an implementation thanks to a process based on models transformations [3,4]. Many models and metamodels have been developed and a design process (figure 1) has been implemented as a set of models transformations. The first transformations that were defined introduced the general architecture of the implementation. In this paper we want to describe the way we have automated the design choice related to data placement and distribution.

We argue that if models can describe *the product* to develop and metamodels the abstractions and constrains that are used and reused to define models, models transformations can be used to describe *the process* to follow. Hence, we have analysed the different design choices that are open when implementing communication components. Among them, we isolated data placement strategies. We have then look for the abstractions required to describe data placement and elaborated a dedicated metamodel. The validation of this metamodel was made thanks to the definition of some models of data placement such as "centralized placement" "peer-to-peer Pastry placement strategy" or "peer-to-peer Chord placement strategy".

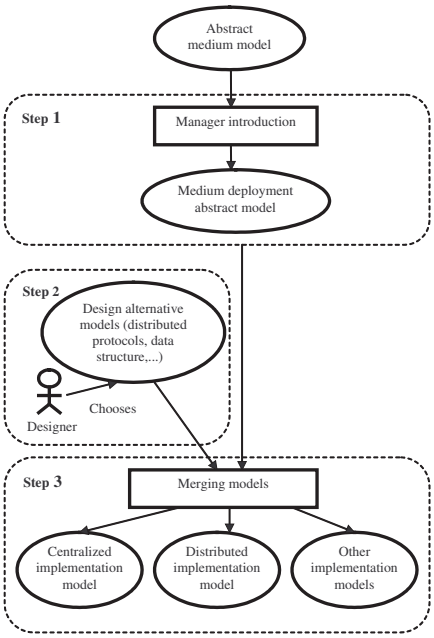

Fig. 1. A view of a medium full design process

In order to apply these design choices we had to define a model transformation that was compatible with or target: communication components. The transformation was hence defined using both metamodels: the communication component one and the data placement one. This choice guarantees the transformation to be reusable since applicable to all models that conforms: any communication component model or any data placement model. Finally, this approach ensures also extensibility since new data placement models - if conforming - could be added.

The paper is organized as follows. The next section summarizes the definition and the deployment target of communication components in order to ensure a better understanding of metamodels and transformation. Section 3 presents our approach defining the implementation parts of data distribution issues as a sequence of transformations in a communication component design process. Section 4 presents some related works. We conclude the paper in section 5 with some perspectives of this work.

2 Communication component: medium

Definition. A medium is a special component which implements any level communication protocol or system. A medium can implement, for example, a consensus protocol, a multimedia stream broadcast or a voting system. A medium includes classical component properties such as explicit interface specification, reusability or replaceability, but a medium is not a unit of deployment. A communication component is a *logical* architectural entity built to be *distributed*. An application is the result of inter-connecting a set of components and mediums. This is particularly interesting as it would allow the separation of two concerns: local concerns described by components and communication concerns described by mediums.

Example. As an illustration, we reuse the example published in [3] of an airline company with travel agencies located worldwide. A medium can implement the reservation system and offer services to initialize information on seats, to reserve seats and to cancel reservations. A reservation application can then be built by inter-connecting the reservation medium and components representing the company and the agencies as illustrated in figure 2

Deployment target In the previous section, we saw that, at the abstract level, the medium is represented by a single software component. The goal of the design process is to make the distribution of this abstraction possible. The single software component which represents the medium at the abstract level is split into small implementation components called *role managers*. Each *role manager* is locally associated with a local component and the medium becomes a logical unit composed of all the *role managers*. From a local point of view, each *role manager* implements the services used by its associated component. From

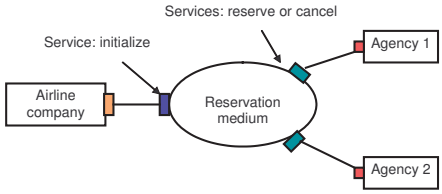

Fig. 2. An example of communication component: reservation medium

a global point of view all the *role managers* communicate through middleware and cooperate to realize all the medium services.

Thus, at the deployment level the single software communication component which represents the medium at the abstract level disappears completely and the medium becomes an aggregation of distributed *role managers*. The data manipulated by the medium at the abstract level are distributed between role managers.

The next section presents our approach to implement data distribution features as a sequence of transformations. We note that the definition of technical details related to elements which are used to ensure data distribution and access services is out of the scope of this paper.

3 Our approach

3.1 Analysis

Identifying the source and the target of the transformation

Identifying the source. Managers introduction is out of the scope (step 1 in figure 1) of this paper. Thus, in the context of this paper, the source of the transformation is a medium deployment abstract model in which: (1) a *manager* is associated to each role; (2) each *manager* implements all the service offered by the medium to its associated role; (3) each role is separate from the other elements constituting the medium. (4) the medium is defined by the aggregation of *managers* as illustrated in figure 3 for the reservation medium.

Identifying the target. The target of the transformation is a medium implementation model in which: (1) a *manager* is associated to each role; (2) each *manager* implements all the service offered by the medium to its associated role; (3) the entity representing the medium in the source specification is deleted; (4) the medium is defined by the aggregation of *managers*; (5) each data managed by the medium in the source specification is distributed between *managers*. The description of the target does not provide design alternatives that will be used to implement data distribution features. It just specifies the set of constraints that each final implementation model of the medium should satisfy. Both source and target descriptions are detailed in [5].

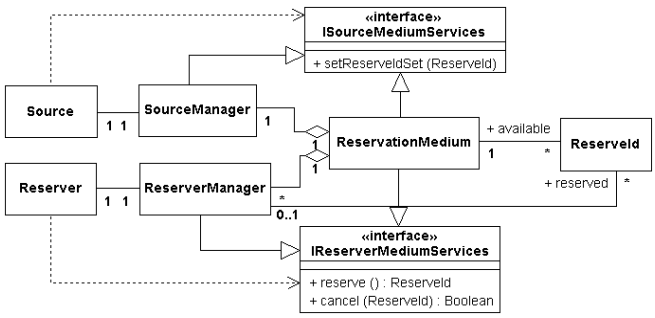

Fig. 3. Structure of the reservation medium after manager introduction

Identifying and separating design alternatives. We decompose the problem of implementing data distribution issues in three design alternatives for the moment. Others design alternatives related to data distribution such as synchronization or context adaptation aspects can be added in future works.

First design alternative: data distribution topology choice. It consists in specifying the set of managers that can participate to the distribution of each data and those that can only access to it.

Second design alternative: distributed protocol choice. It specifies the distributed protocol (*Chord [6]*, *Pastry [7]*, ...) that will be used to implement the distribution strategy of each data.

Third design alternative: distributed protocol implementation algorithm. The last step is the choice of the algorithm that will be used to implement each distributed protocol services. As an illustration, in the case of the *Chord* protocol, the designer can choose the algorithm proposed by the *MIT* [8] or the algorithm proposed the *MACEDON* [9] framework to implement the protocol.

3.2 Automation

In this section, we sketch out metamodels and transformations that we use to describe and automate the introduction of design alternatives identified in the previous section in the medium deployment abstract model.

Metamodelling We define a different metamodel for the source, the target and distributed protocols in order to ensure a better understanding of metamodels. Each metamodel is specified with to elements: a UML class diagram describing the generic structure of the concept and a set of OCL specifications describing the properties of the concept which cannot be expressed in the class diagram. In sake of brevity, we only show the generic structure of each metamodel in this paper. The full definition of all the metamodels is available in [10].

Medium deployment specification metamodel. Figure 5 shows the generic structure of a medium during the deployment. A *manager* (*</RoleName>Manager*) is associated to each role (</RoleName>). Each *manager* implements the interface of service offered by the medium (*I<RoleName>MediumServices*) to its associated role and the medium (*<MediumName>Medium*) is defined by the aggregation of *managers*.

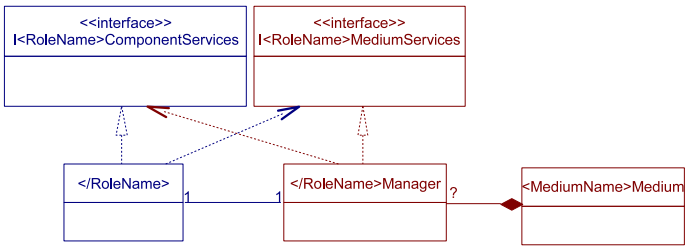

Fig. 4. Generic structure of a medium at the deployment level

Medium implementation specification metamodel. Figure 5 shows the generic structure of a medium during the implementation. The medium class disappears completely and the medium data are distributed between managers. The gray color delimits our metamodel of a distributed data. Each distributed data is represented by two elements. The first element (*DataManager*) ensures the data distribution services and the second (*DataObject*) element ensures the data access services.

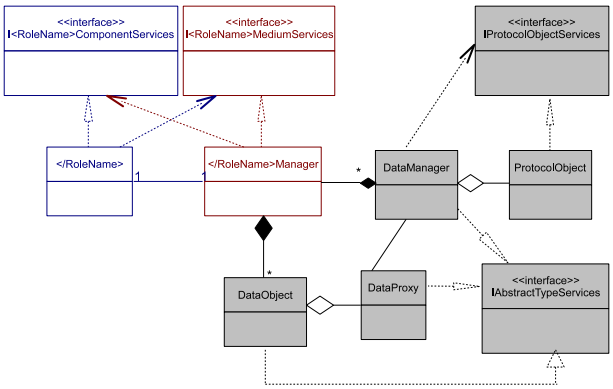

Fig. 5. Generic structure of a medium at the implementation level

Distributed protocol metamodel. We define a distributed protocol by a set of objects called *ProtocolObject* (figure 6). A *ProtocolObject* is an object that can execute the behaviour of a distributed protocol. Each *ProtocolObject* is implemented by a specific algorithm (*ProtocolObjectAlgorithm*). The main goal of the distributed protocol metamodel consists in defining a common interface for all distributed protocols that will be used in the context of mediums. Such interfaces are proposed in [11,9,7]. The interface *IProtocolObjectServices* exported by the distributed protocol definition metamodel is similar to the interface defined in [11]. This interface defines services for three main distributed application abstractions: the DHT (Distributed Hash Tables), DOLR (Decentralized Object Location and Routing) and the CAST (group anycast/multicast). The interface *IProtocolObjectServices* offers the following services: *route* (to route a message), *forward* (to forward a message), *deliver* (to deliver a message), *join* (to join the distributed application) and *leave* (to leave the distributed protocol).

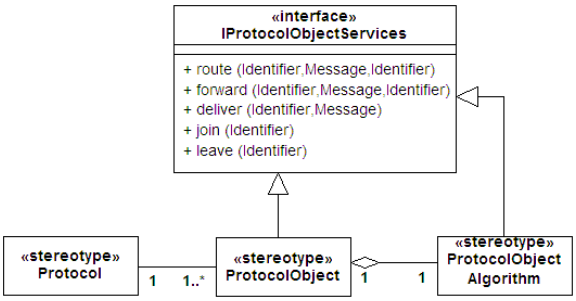

Fig. 6. A view of the distributed protocol specification metamodel

A definition of a distributed protocol model which conforms to the distributed protocol metamodel consists in: (1) the description of each *ProtocolObject* and at least one of its implementation algorithms and (2) the implementation of each service offered by the *IProtocolObjectServices*. We illustrate the definition of a *Chord* protocol model in [12].

3.3 Model transformation

Principle. The entry point of the transformation is a well defined medium deployment abstract model (figure 3) in this paper. In the full design process of the medium, we perform a first transformation in order to transform this model into an abstract implementation model in which all the UML associations between the medium class and the medium data classes are replaced by the appropriate abstract types specified by the designer [13]. In the case of the reservation medium for example, the available reservation identifiers (**available**

property in figure 3) can be represented by a **list**. Since we haven't discuss abstract types choice in this paper, this transformation will not be described in this section. After the introduction of abstract types, we perform five successive transformations in order to introduce data distribution topologies, distributed protocols and distributed protocol algorithms in the abstract implementation model of the medium. These transformations leads to a generic implementation model implementing the actual data distribution variant. The designer completes this generic implementation model by defining each offered service actual implementation algorithm to produce the final implementation model of the medium. The following example describes a data distribution variant that will be used to illustrate transformations in remain of this section.

An example of data distribution variant. We will use the reservation medium to illustrate transformations in this section. We suppose that the available reservation identifiers (**available** property) are represented by a list. The goal is to distributed this list between managers associated to agencies (*ReserverManager*) using the *Chord* protocol. The *manager* associated to the airline company (*SourceManager*) can only access to the list. The Chord protocol will be implemented by the *MIT* algorithm. These information are defined in a medium decision model in the full implementation process in order to allow the automate the execution of transformations [13].

T_1. *Introducing each data distribution topology in the abstract implementation model of the medium.* The input model of this step is the medium abstract implementation model obtained after abstract types introduction. We aim at introducing each data distribution topology in this model. We define a transformation based on the medium deployment and implementation metamodels and the distributed data metamodel for this purpose. This transformation leads to an abstract implementation model of the medium in which : (1) a *DataManager* is associated to each *manager* participating to each data distribution and (2) a *DataObject* is associated to each *manager* accessing to each distributed data. Here is an informal summarize of its main operations.

```
Preconditions:
 1.Verify if each distribution node is defined in the medium model.
Actions:
  for each data managed by the medium class:
   1.Create and associate a generic DataManager object to each manager
     participating to data distribution.
   2.Create and associate a generic DataAccess object to each manager
      using the data
Postcondition:
  Verify if a DataManage and/or a DataObject is associated to each
  manager according to the implementation variant.
```

As an illustration, in the example of the reservation medium, the transformation associates a *ListDataManager*to *ReserverManager* and a *ListObject* to both *SourceManager* and *ReserverManager* (figure 7).

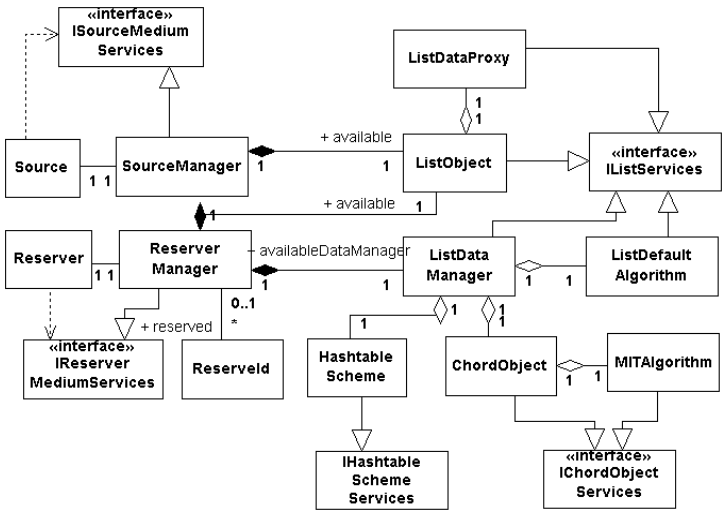

Fig. 7. A view of the reservation medium after T_1, T_2 and T_3

T_2. *Introducing distributed protocol in the abstract implementation model of the medium* In this step we define another transformation based on the same metamodels as T_1 to introduce the distributed protocol that will be used to ensure each data distribution strategy in the abstract implementation model generate by T_1. Its operations can be summarize as follows.

```
Preconditions:
 Verify if the model of each distributed protocol conforms to
 the distributed protocol metamodel
Actions:
1. Create and associate a generic ProtocolObject to each DataManager
   according to the distribution variant.
Postcondition:
 Verify if a generic ProtocolObject is associated to each DataManager
   according to the distribution variant.
```

T_3. *Introducing distributed protocol algorithms in the abstract implementation model of the medium* This transformation perform the following operations in order to introduce the implementation algorithm of each distributed protocol object in the model generated by T_2 as illustrated in figure 7.

```
Preconditions:
  Verify if each distributed ProtocolObject implementation algorithm is
  well defined according to distributed protocol metamodel.
Actions:
 Create and associate a generic ProtocolObjectAlgorithm to
```

```
 each protocol object according to the distribution variant.
Postcondition:
  Verify if a generic ProtocolObjectAlgorithm is associated
  to each ProtocolObject according to the distribution variant.
```

T_4. *Generating abstract methods implementation algorithms in the abstract implementation model of the medium* In this step, we define a transformation to generate a default algorithm in order to implement each abstract method contained in the abstract implementation model produced by T_3. Here is an example of piece of code generated by this transformation in order to implement the add primitive of the **list** for the reservation medium example.

```
class ListObject inherits IListServices
{    ...
    method get (index : Integer ): Object from IListServices is do
        if (self.dataProxy != void) then
             then result := self.dataProxy.get(id) end
        end end    // suite du code}
class ListDataProxy inherits IListServices
{    ...
    method get (index : Integer ): Object from IListServices is do
        var dataManager : ListDataManager init getListDataManager()
        if (dataManager != void) then
             then result := dataManager.get(id) end
        end end
//other instructions }
```

T_5. *Configurating the medium* The previous step leads to an abstract implementation model of the medium in which all the generic elements needed to provide each data distribution and access services are well defined. In this step, we define a last transformation based on the same metamodels as the previous transformation in order to instantiate and associate the appropriate value to each generic element. Four generic operations named *managerConnection, managerDisconnection, initialization and termination* are defined in [5] in order to associate a specific behaviour to a manager during its connection, disconnection, initialization and termination. The last transformation redefine these operations in order to reach its goal. Here are an informal summery of the main operations performed by the last transformation an example of generated code for the reservation medium.

```
Precondition:
 Verify if all abstract methods are implemented in the input model.
Actions:
 1.Redefine the managerConnection, managerDisconnection, initialization
   and termination operations in each Manager class.
 2.for each distributed data:
   2.1.Generate instructions in managerConnection operations to
       instantiate the appropriate protocol objects and protocol object
```

```
      algorithms according to the design choice
   2.2.Set the data manager and the data object values
  2.1.Generate instructions in managerDisconnection operations to
      disconnect protocol objects
  2.1.Generate instructions in initialization operations to init protocol
      objects
  2.1.Generate instructions in termination operations to terminate protocol
      objects.
Postcondition:
 Verify if the output is a well medium implementation specification model
 according to the medium decision model and the medium implementation
 specification metamodel
```

```
class ReserverManager inherits IReserverMediumServices
{....operation managerConnection() is do ....
       available := ListObject.new
       available.dataProxy := ListDataProxy.new
       availableProtocolObject := ChordProtocolObject.new
       availableProtocolObject.protocolObjectAlgorithm := MITAlgorithm.new
       availableDataManager := ListDataManager.new
       availableDataManager.protocolObject := availableProtocolObject
       availableDataManager.listDefaultAlgorithm := ListDefaultAlgorithm.new
          // other instructions     end }
```

Transformation definitions platform. Transformations are implemented, testing and executed in Kermeta [14] platform. Each metamodel is implemented by two Kermeta files. The first file implements all the structural aspects of the metamodel. The second file implements all the properties of the metamodel. It is then possible to check if a specific model is conform to the metamodel in which it is defined. Each transformation is implemented by three Kermeta files. The first file implements the preconditions, the second file implements the operations and last file implements the postconditions of the transformation. A full definition of metamodels and transformations in Kermeta is available in [10]

4 Related works

Most methodologies are informally described. They suggest a process which, in the most formalized cases, rely on contracts [15] or mathematical refinements like the B-method [16]. B defines a language and a refinement methodology. It is an algebraic specification language that is supported by tools that help refining specification safely. Each step of the process generates proof requirement the developer has to demonstrate, either manually or automatically. Some critical systems were developed in B (in 1998 the control system of line 14 of the Parisian subway was fully developed and proved in B). Our approach is more empirical and uses the so-called "semi formal" approach. It may be easier to learn and may tackle different kind of design problems such as distribution. We do not try to prove design steps, but just to automatize them and give enough confidence in the transformations thanks to pre and post-conditions.

In a recent paper, H. Sneed [17] criticizes the model driven approach. He argues that model-driven tools (1) magnify the mistakes made in the problem definition, (2) create an additional semantic level to be maintained, (3) distort the image of what the program is really like, (4) complicate the maintenance process by creating redundant descriptions which have to be maintained in parallel,(5) are designed for top-down development that creates well-known maintenance problems. These drawbacks are mainly associated with tools. All these criticisms have already been raised when assembly was replaced by high level programming languages. We agree tools are not mature. Our experiment shows that transformations may help make explicit the process and simplify the maintenance, if models are defined well enough.

Other experiments [18] tend to prove that model composition (hence a bottom-up approach) is possible. This compositional approach looks like Aspect Oriented Modelling [19]. This approach recommends to separate concerns and offers an operation of weaving that composes/weaves each concern with the functional specification. Our approach differs since the "weaving" operation we use is a transformation that is adapted to the kind of concern composed. Instead of using a universal weaving operation we propose a more flexible approach (but less re-usable) were a balance may be found between the metamodel definition of the concern and its composition operation implemented as a transformation.

Model transformations are widely used on UML models. Most of them cover a small part of the development life cycle. Some transformations are dedicated to code generation. They usually produce the skeleton (structural part) of the source code that has to be completed manually. Another current use is applying design patterns [20]. Once again, the structural part is rather well implemented[1], but the collaboration one is still research in progress.

5 Conclusion

This paper shows how models transformations can be used to describe the implementation process of data distribution issues in a distributed software component. To do this we have defined metamodels that capture the required concepts of data distribution. We also have realized a sequence of models transformations that weave a variant of data distribution design choice into the model of a distributed component.

The transformations describe the process of introducing actual design alternatives models in the specification of the component. The approach makes explicit the data distribution implementation process. We argue it is very interesting in the sense that it facilitates the traceability (the sequence of transformations), the reuse (applicability of transformations on many different models) and the evolution of the full process (adding more variant models).

[1] Patterns purists would say that patterns are not dedicated to be automatically applied. In the absolute, we agree, but why not consider applying patterns in well defined contexts?

As an illustration, we have applied our approach to implement data distribution in the context of mediums. We have described a set of transformations and metamodels that can be used to introduce distributed protocols.

But the concern of data distribution is only one step in a larger design process. We have described an approach based on the definition of a sequence of design concerns. As an example we have selected the choice of an abstract type for collection of data, the choice of distribution strategies (described in this paper) and the choice of data representation format.

Transformations are implemented, tested and executed with the Kermeta platform. The implementation of the full design process relies on 6 metamodels and 5 main transformations. Each metamodel is implemented by two Kermeta files. The first file implements all the structural aspects of the metamodel. The second file implements all the properties of the metamodel. It is then possible to check if a specific model is conformed to the metamodel in which it is defined. Each transformation is implemented by three Kermeta files. The first file implements the preconditions, the second the operations and the last file the transformation postconditions. We also provide a library containing some abstract type (list, set, bag, ...), distributed protocol (chord, pastry, ...) and data representation format models (hashtable, array, matrix, ...).

Transformations can be used to implement any abstract type, distributed protocol and data representation format models which conforms to our metamodels in any well defined medium initial specification model. As an illustration, we have used transformations to automatically derive various centralized and distributed implementation variants of the reservation medium presented in this paper and two other mediums: a voting medium and a message broadcast medium.

The actual generated implementation models of mediums are not fully executable in the sense that they to not provide a full executable code of distributed protocols. They just call protocol APIs to ensure that all distributed feature are well implemented. Thus, in short time, our main perspective is to build a middleware in order to allow the execution of the generate models in conjunction with existing executable distributed protocol frameworks such as MACEDON. We also aim at defining other design concerns metamodels and models to enrich our library. After that, we aim at extending transformations to define auto-adaptable mediums that embed many variants and that could change there internal deployed structures according to environment evolutions.

References

1. Cariou, E., Beugnard, A.: The specification of UML collaboration as interaction component. In J.M. Jézéquel, H. Hussmann, S. Cook, ed.: ¡¡UML¿¿ 2002 – The Unified Modeling Language. Volume 2640 of LNCS., Dresden, Germany, Springer Verlag (September 30 - October 4, 2002) 352 – 367
2. Matougui, S., Beugnard, A.: Two ways of implementing software connections among distributed components. In: International Symposium on Distributed Objects and Applications, Agia Napa, Cyprus (2005)

3. Cariou, E., Beugnard, A., Jézéquel, J.M.: An archictecture and a process for implementing distributed collaborations. In: The 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland (2002)
4. Kabore, E., Beugnard, A.: Conception de composants répartis par transformations de modèle. In: Journées de l'Ingénierie Dirigée par les Modèles, Toulouse, France (2007) 117–131
5. Cariou, E.: contribution un processus de rification d'abstraction de communication. Thse de doctorat, Universit de Rennes 1 (2003)
6. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: ACM SIGCOMM Conference, San Diego (2001)
7. Rowstron, A., Drusche, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM Middleware. (2001)
8. Massachusetts Institute of Technology: lsd. http://www.pdos.lcs.mit.edu/chord/ (2004)
9. Rodriguez, A., Killian, C., Bhat, S., Kostic, D., Vahdat, A.: Macedon: Methodology for automatically creating, evaluating, and designing overlay networks. In: USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004). (2004)
10. Kaboré, E.: Metamodel definitions. http://stockage.univ-brest.fr/ kabore/ (2008)
11. Dabek, F., Zhao, B., Drushcel, P., Kubiatowicz, J., Stoica, I.: Towards a common api for structured peer-to-peer overlays. In: 2nd International Workshop on Peer-to-peer Systems (IPTPS'03). (2003)
12. Kabor, E., Beugnard, A.: On the benefits of using model transformations to describe components design process. In: Twelfth International Workshop on Component-Oriented Programming (WCOP'2007), at ECOOP'2007, Berlin, Germany (Juillet 2007)
13. Kaboré, E., Beugnard, A.: Automatisation dun processus de conception par transformations de modles. L'Objet (2008) to appear.
14. Muller, P.A., Fleurey, F., Jzquel, J.M.: Weaving executability into object-oriented meta-languages. In Briand, S.K.L., ed.: Proceedings of MODELS/UML'2005. LNCS, Montego Bay, Jamaica, Springer (2005)
15. D'Souza, D., Wills, A.C.: Objects, Components and Framework with UML: The Catalysis Approach. Addison-Wesley (1998)
16. Abrial, J.R.: The B-Book, Assigning Programs to Meanings. Number ISBN 0521496195. Cambridge University Press (1996)
17. Sneed, H.M.: The drawbacks of model-driven software evolution. In: Workshop on Model-Driven Software Evolution, IEEE - CSMR 2007 11th European Conference on Software Maintenance and Reengineering "Software Evolution in Complex Software Intensive Systems", Amsterdam, the Netherlands (2007)
18. Muller, A., Caron, O., Carré, B., Vanwormhoudt, G.: On some properties of parametrized model application. In: First European Conference on Model Driven Architecture - Fondations and Applications (ECMA-FA'05). Volume 3748 of LNCS., Springer (2005) 130–140
19. Mens, K., Lopes, C., Tekinerdogan, B., Kiczales, G.: Aspect-oriented programming. Lecture Notes in Computer Science **1357** (1998)
20. Sunyé, G., Guennec, A.L., Jézéquel, J.M.: Design pattern application in UML. In Bertino, E., ed.: ECOOP'2000 proceedings. Volume 1850 of Lecture Notes in Computer Science., Springer Verlag (2000) 44–62