Towards a Multi Metamodelling Approach for Developing Distributed Healthcare Applications

Fazle Rabbi[1,2], Yngve lamo[1], Ingrid Chieh Yu[2],
Lars Michael Kristensen[1]
[1] Bergen University College, Bergen, Norway
fra@hib.no, yla@hib.no, lmkr@hib.no
[2] University Of Oslo, Oslo, Norway
fazlr@student.matnat.uio.no, ingridcy@ifi.uio.no

Abstract

Model Driven Engineering (MDE) uses formal methods to build mathematically rigorous models of complex systems. Metamodelling plays an important role in MDE as it is used to specify domain specific modelling languages. However, the potential of metamodelling has not been fully explored. Current approaches of MDE are often at a low level of abstraction and lack domain concepts for specifying behavior. In previous work, we proposed a multi metamodelling approach that captures the complexity of systems by using a metamodelling hierarchy, built from individually defined metamodels, each capturing different aspects of a healthcare domain. In this paper, we focus on modelling distributed healthcare applications and present an example from the healthcare domain. We address certain modelling aspects related to distributed applications such as process modelling, using message passing communication, and coordination of processes and resources.

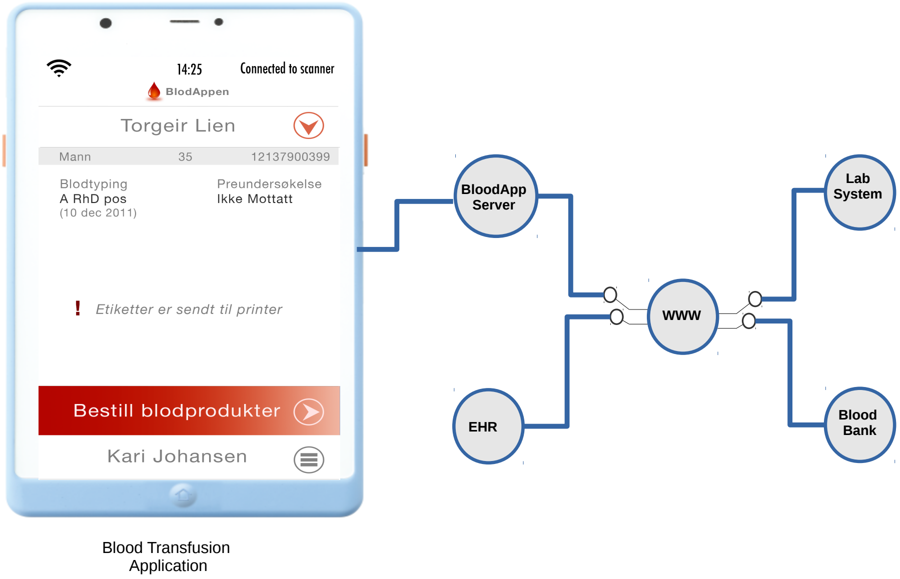
1 Introduction

Healthcare systems are complex in many dimentions as there are many resources involved that need to be coordinated and in most cases the processes in a healthcare applications are safety critical. Typically a healthcare system consist of many different applications providing a wide range of services. Coordinations among these applications are very important for delivering high quality healthcare. However, in many situations the coordination is performed manually which is not scalable and which can be improved by integrating applications. In this paper, we study a formal approach for metamodelling of distributed healthcare applications. Since hospitals operate on different settings and platforms, a single model or modelling language is not well-suited to fit all aspects of the modelling requirements. We therefore propose the use

This paper was presented at the NIK-2015 conference; see http://www.nik.no/.

of multi metamodelling which supports the development of domain specific modelling languages to accomodate different situations. As an example we discuss the modelling of a blood transfusion application that we are developing in collaboration with Helse Bergen and Helse Vest IKT. The goal of the project is to develop an application for safe blood transfusion. We are modelling the process flow and developing the data models as well as modelling the interaction between different subsystems.

Hospitals in developed countries are taking initiative to use handheld devices targetting increased use of information and communication technology into healthcare processes. The use of handheld devices allow caregivers to interact with systems while they are beside a hospital bed providing care to patients. In general, a blood transfusion application running on a handheld device need to communicate with different systems such as the Electronic Health Record (EHR), the laboratory system, and the blood bank. Figure 1 shows a screenshot of a blood transfusion application and its collaboration with other healthcare systems. It is very common that these systems are provided by different vendors and they are not necessarily built on a common data model. The modelling and simulation of a blood transfusion application therefore requires to facilitate the modelling and integration of heterogeneous applications. Multi metamodelling has the potential to deal with this modelling requirement.


14:25 Connected to scanner

Torgeir Lien

Mann 35 12137900399

Blodtyping
A RhD pos
(10 dec 2011)

Preundersøkelse
Ikke Mottatt

! Etiketter er sendt til printer

Kari Johansen

Figure 1: A screenshot of a blood transfusion application and its collaboration with other heathcare systems

A typical hospital scenario for blood transfusion is as follows: In order to reduce human error and increase patients safety, the blood transfusion application must read patient's identification from the wristband. A nurse can use the application to get information about the patient's blood type from the patient's EHR. If a patient's blood type is not known, patient's blood sample is collected to determine the blood type. A tag is printed and attached to the blood sample before sending it to the laboratory. Nurses can order blood products from the blood bank with blood type or tag number from patient's blood sample. The laboratory engineers perform screening tests and send test results to the blood bank. The

Blood bank then delivers blood products to the appropriate departments. Before any blood transfusion is performed on a patient, the patient's identification is matched with the label from the blood product and verified by two nurses. During the blood transfusion if any complication is found then these are recorded and sent to the blood bank. We take this scenario into consideration and use multi metamodelling for designing a distributed blood transfusion application.

In this paper, we present a workflow that integrates the processes of the distributed application. The proposed approach is based on the principles of the Diagram Predicate Framework (DPF) [1] [2] and Dynamic logic [3]. DPF provides a formal diagrammatic approach to multilevel metamodelling based on category theory and model transformations which provides an abstract visualization of concrete constraints. We based our approach on multilevel metamodelling since it offers a clean, simple and coherent semantics for metamodelling [18]. DPF can be used for structural and behavioral modelling of a system. A workflow model in DPF represents an abstract process model which may be used for the analysis of a system. In order to specify the details of a process, such as the specification of a process, we propose the use of dynamic logic. Dynamic logic is used to provide dynamic behavior to the processes. Dynamic logic has strong connections with classical logic and provides a formal reasoning for programs. The use of dynamic logic for specifying process description of DPF models provides us with a formalism to construct rich metamodelling languages. The overall approach for multi metamodelling of distributed applications is depicted in Figure 2. The figure illustrates the use of different metamodelling hierarchies for modelling the process model, and entity models: processes in the workflow model are annotated with their location of execution; messages are formalized as typed graph; processes located in the same execution point share entity models; dynamic logic is used for writing process specifications.

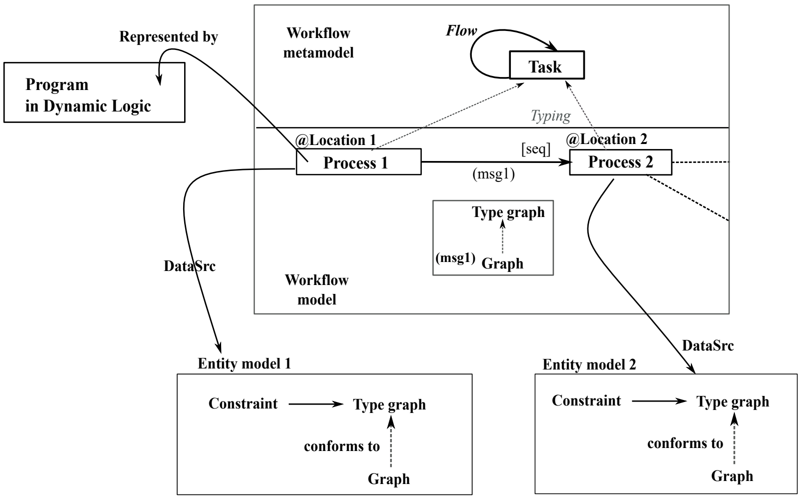


Figure 2: Overall approach for multi metamodelling of distributed applications

2 Multilevel metamodelling

Models at each level in a metamodelling hierarchy are specified by a modelling language at the level above and conform to the corresponding metamodel. This is known as multilevel metamodelling [18]. With DPF we construct a metamodelling hierarchy in which a model at any level can be considered as a metamodel wrt.

the models at the level below it. In the DPF approach, models at any level are formalised as diagrammatic specifications which consist of type graphs and diagrammatic constraints. DPF is a language independent formalism for defining metamodelling hierarchies. Constraints are added into a DPF model by graph homomorphism and the semantics of the constraints are specified in logic (e.g., OCL, Alloy).

In this paper, we use DPF to model the structure and behaviour of a blood transfusion application (*Blood-App*), patient's electronic health record (*EHR*), laboratory system (*LabSys*), and a blood bank system (*Blood-Bank*). These systems may have different entity models to store information. While the *EHR* keeps information about patients, caregivers and departments, the *LabSys*, *Blood-Bank*, and the *Blood-App* do not require all this information and therefore their entity models are different. For simplicity we have used a common metamodel and a common vocabulary in the entity models. Figure 3 shows four DPF models with their metamodels representing the entity models of the *EHR*, *LabSys*, *Blood-Bank*, and *Blood-App*. The graphs represent the structure of the models; constraints are added into the structure by predicates. Table 1 shows two predicates that we have used for constraining the entity models in Figure 3. Each predicate has a name (p), a shape graph, a visualisation, and a semantic interpretation. The semantic of a predicate is provided by a set of instances. In the *EHR* entity model, the [*composite*], and [$injective$] predicates are used to constraint the model. The *EHR* entity model have following three constraints: *a Caregiver who works for a Ward must have access to the Patient's information who are admitted into the same Ward*; *a Caregiver who works for a Ward must work for the controlling Department*; *every Patient has a unique patientID*. All the instances of the *EHR* entity model must satisfy these constraints.

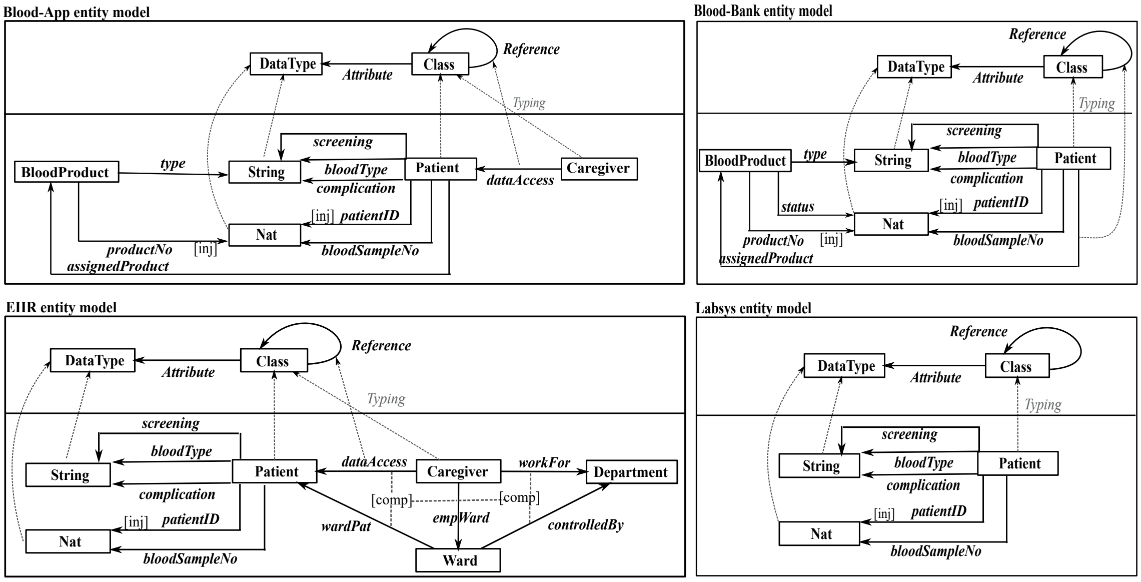
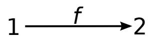
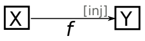
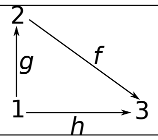
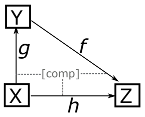


Figure 3: Entity models of *Blood-App*, *Blood-Bank*, *EHR*, and *LabSys*

A DPF metamodelling hierarchy consists of (a possible stack of) metamodels, models, and instances of models. A metamodel specification determines a modelling language, the specification of a model represents a software system, and the instances of models typically represent possible states of a system.

Table 1: Predicates of a sample signature

p	Shape graph	visualisation	Semantic Interpretation
[injective]			Every instance of Y is the image of at most one instance of X. Formally, $\forall x, x' \in X$: $f(x) = f(x')$ *implies* $x = x'$,
[composite]			For each instance of $(g; f)$, there exists an instance of h. Formally, $\forall x \in X : \bigcup \{f(y) \mid y \in g(x)\} \subseteq h(x)$

3 Modelling distributed systems

Rutle et al. proposed a metamodelling approach for behavioral modelling in [4] and developed a workflow modelling langauge named DERF using coupled model transformation rules. For modelling distributed systems we extend the DERF workflow modelling language with the concept of process location, message passing communication, coordination of processes with entity models, and process refinement with dynamic logic.

In general, a workflow or process modelling language requires different types of routing predicates to control the flow of execution among processes. The state of a workflow is determined by the states of task instances (i.e., processes) in DERF. Each task instance is annotated with a predicate from [*Running*] and [*Disabled*] (in short <*R*>, <*D*>, resp.) to indicate its state [1]. A task instance annotated with the predicate [*Running*] indicates that the task instance is in the executing state and may transit to the disabled state (e.g., become annotated with [*Disabled*]) when it has finished its execution.

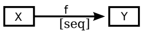
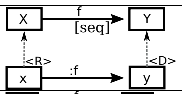
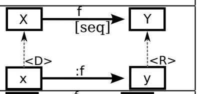
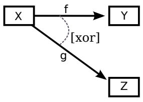
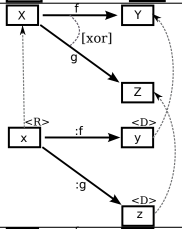
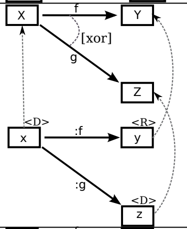
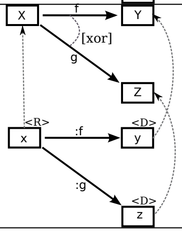
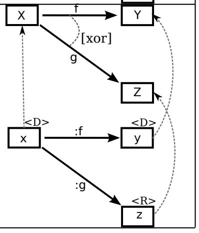
Table 2 shows a set of selected routing predicates and transformation rules (adapted from [4]) that can be used to construct a workflow model. The [*sequence*] predicate indicates a sequential execution order of tasks. The transformation rule for the sequence also illustrates a rule that changes the annotation of a task instance x from <*R*> to <*D*> and the annotation of a task instance y from <*D*> to <*R*>, respectively. The [*xor_split*] predicate specifies that exactly one of the two flows must be followed.

We model the behaviour of a distributed system by a workflow where the processes belong to different systems. We annotate processes with system names to indicate the location where the processes are executing. For example, we use *@App*, *@EHR*, *@LabSys*, *@BloodBank* to indicate the location of execution of the processes. Here *@App* denotes the blood transfusion application, *@EHR* denotes the patient's EHR, *@LabSys* denotes the laboratory system, and *@BloodBank* denotes the blood bank. Figure 4 shows two processes 'Scan Patients Wristband' and 'Get Patient Info' sequentially connected by the [*sequence*] routing predicate. The 'Scan Patients Wristband' process has been annotated with *@App* which indicates that the process executes at the *Blood-App*. Figure 4 also shows two instances ':Scan Patients Wristband' and ':Get Patient Info' of processes 'Scan Patients Wristband' and 'Get Patient Info'. The states of the process instances are represented by the annotations <*R*>, and <*D*>.

We model the message passing communication between processes by predicates. A message predicate ([$Msg(m)$]) can be used in a DPF model by graph homomorphism. Figure 4 shows a message passing communication between

[1] Here we use two states for task instances rather than four states ([*Disabled*], [*Running*], [*Enabled*], [*Finished*]) as proposed in DERF [4].

Table 2: Routing predicates and transformation rules for workflow modelling

p	Visualization	Coupled transformation rules	
		LHS	RHS
[sequence]			
[xor_split]			
			

two processes 'Scan Patients Wristband' and 'Get Patient Info'. When a [*Msg*(*m*)] predicate is used in a model, the parameter *m* is bound to a typed graph. For instance, Figure 4 shows that a natural number ($x : Nat$) is passed from 'Scan Patients Wristband' process to 'Get Patient Info'. During the execution of the workflow, the variable *x* will be replaced by a natural number.

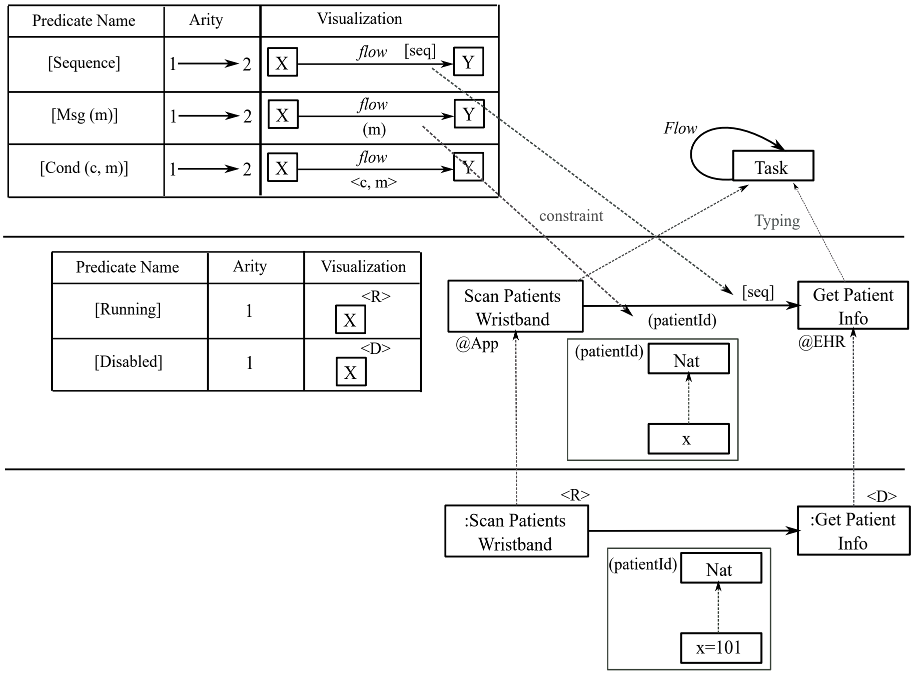


Figure 4: Tasks sending and receiving messages

Figure 5 shows the workflow for the blood transfusion scenario using the routing predicates from Table 2. To order blood products for patients, their wristband is scanned by the *Blood-App*. The scanned ID is sent to the *EHR* where patient's information is retrieved and sent back to the *Blood-App*. If patient's blood type is not found, a tag is printed to put on patient's blood sample, otherwise an order is made for the blood product. The outgoing branches of the 'Check blood Type' is using the [$Cond(c, m)$] predicate to specify branching conditions. The condition predicate [$Cond(c, m)$] has a typed message m and a first-order formula c, which is used to specify a branching logic. After collecting patient's blood sample, it is possible to order the blood product by sending the blood sample number. Screening tests are performed before any blood product is finally dispatched from the blood bank. When a blood product reaches the patient's location, the patient's id is matched and verified by nurses. Before starting a blood transfusion, nurses measure patient's blood pressure and heart rate. During the blood transfusion process, if any complication is arised, they are recorded and sent to the blood bank. Patient's blood pressure and heart rate are also measured after completing the blood transfusion.

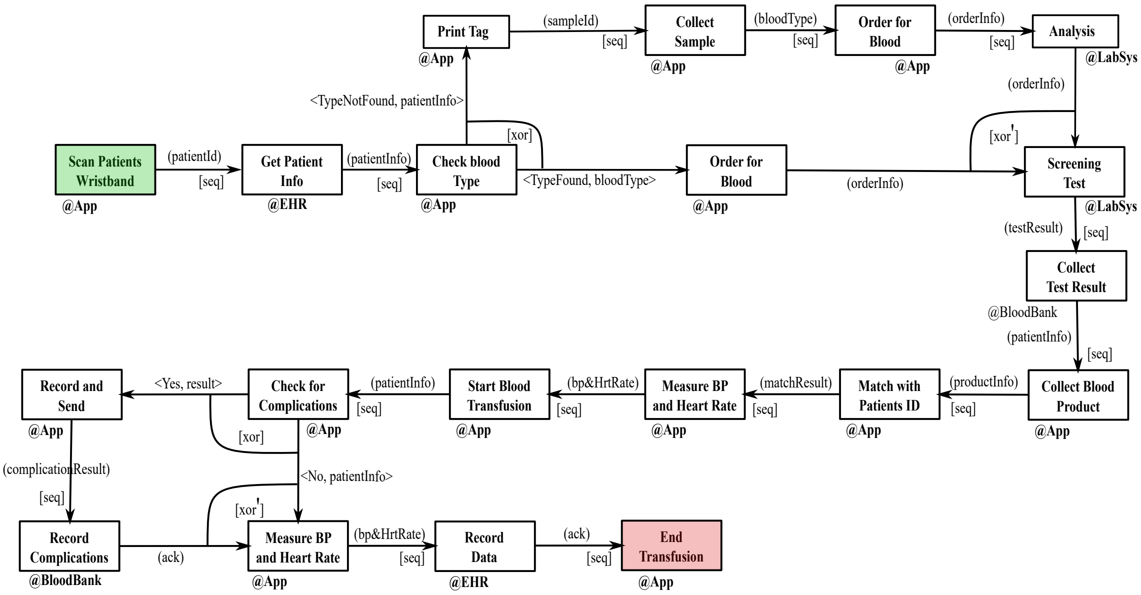


Figure 5: Blood transfusion workflow

4 Message passing between distributed processes

In this section we provide the details of the proposed metamodelling approach for distributed applications. The message passing communication between the processes of a distributed application is explained with a running example from the blood transfusion application. Figure 6 (top) shows three processes from the blood transfusion workflow presented in Figure 5 where the patients wristband is read by the Blood-App, then patient's identification is transferred to the *EHR* to retrieve patient's blood type related information and finally the blood type is checked by the Blood-App. Process instances are coordinated with entity models by the 'DataSrc' relations. The ':Scan Patients Wristband' process instance and the ':Check blood Type' process instance are associated with the same entity model as they belong to the Blood-App application. Figure 6 (bottom) shows sample instances of the entity models. The purpose of a sample instance of a model is to define the scope of a system for simulation. During the execution of the workflow, ':Scan Patients Wristband' changes its state from $\langle R \rangle$ to $\langle D \rangle$; the typed graph of message (*patientId*) is matched with the Blood-App entity model and its sample instance; the variable $x : Nat$ is bound to a value from {101, 201, 301} and therefore (*patientId*) is transmitted to the ':Get Patient Info' process instance. The sample instance of the *EHR* entity model represents the database for patients and caregivers.

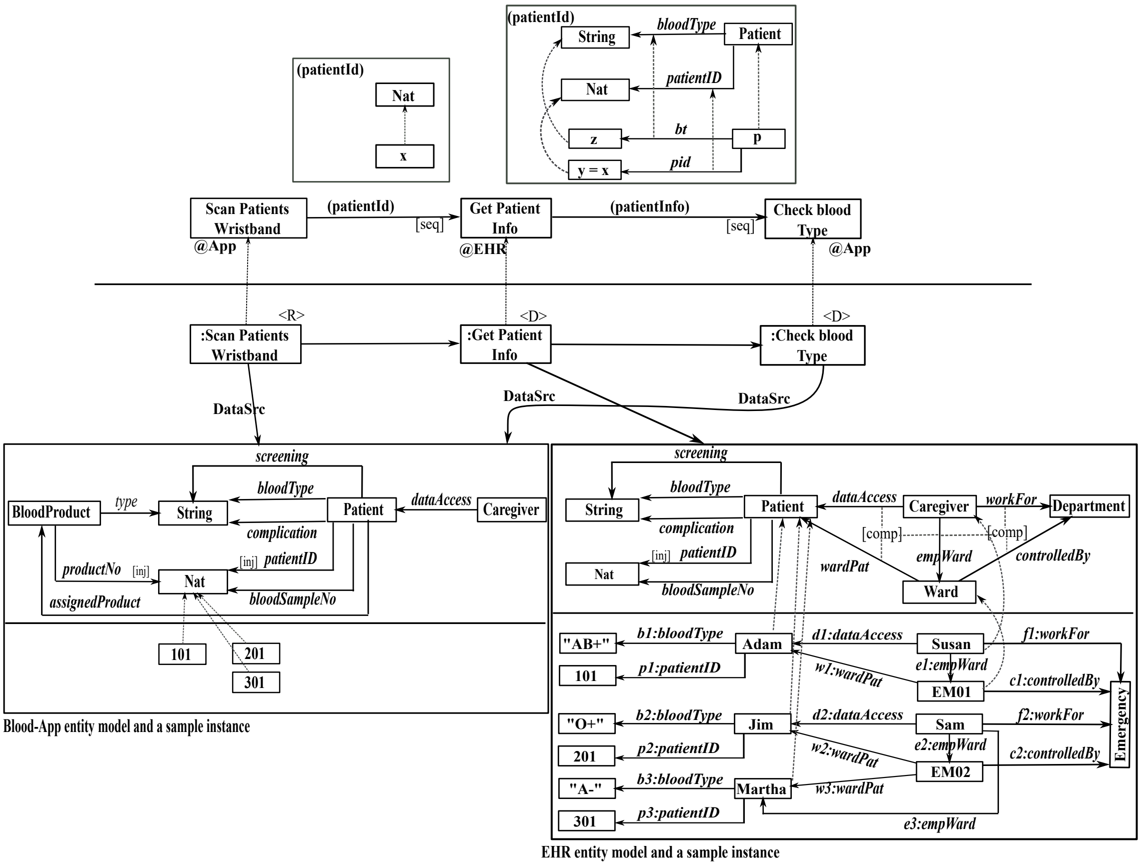


Figure 6: Process instances and their associated entity models

Upon receival of message (*patientId*), the process instance ':Get Patient Info' changes its state from <*D*> to <*R*>. ':Get Patient Info' does not make any change to the patient's database but retrieves patient's information from the database. The typed graph of message (*patientInfo*) is matched with the *EHR* entity model and its sample instance. The matching of (*patientInfo*) retrieves information about patient with patientID '101' and the message is sent to the next process instance. Figure 7 shows a message transfer (i.e., the information being transferred) between ':Get Patient Info' and ':Check blood Type' process instances and their state change.

5 Process description in dynamic logic

We have presented our multi metamodelling approach for modelling distributed applications in earlier sections. The structural and behavioral modelling presented until this point are abstract and we have not detailed what each process instances does in the running state. In a typical software application, processes take input and manipulate data from an entity model based on specific algorithms. Model transformation can be used for data manipulation in a graph-based system. However, to specify a data manipulation algorithm using model transformation rules would be very time consuming. We propose a variant of

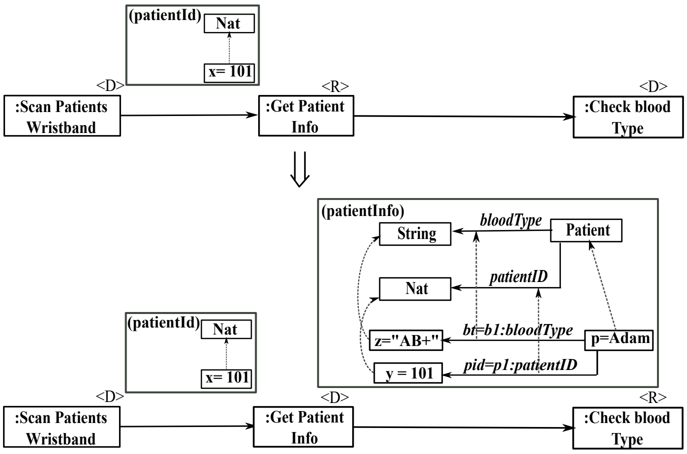


Figure 7: Message transfer between process instances

first-order dynamic logic [3] for specifying the program details of processes. The proposed variant supports updating the domain of computation while the traditional version assumes that the domain of computation is fixed. This support is essential for modelling the behavioral aspects of a system. Atomic programs in this variant are no longer simple *assignment statements* that assign values to variables during the computation, but also changes the interpretation by creating and deleting facts. The atomic program $a = r(term_1, ...term_n)$ inserts a new fact with the predicate symbol r into the interpretation of first-order relations; the atomic program $a = \neg r(term_1, ...term_n)$ deletes an existing fact from the interpretation of first-order relations. In this variant of first-order dynamic logic, states are no longer valuations of a set of variables over the carrier of a first-order structure, but also includes interpretation of the first-order structure $\mathcal{M}$. The execution of an atomic program a changes the state of a system in the following way:

- $a = x := t$ assigns the value of term t to the variable x
- $a = r(term_1, ...term_n)$ updates the interpretation of $\mathcal{M}$ by adding a new fact $r(term_1, ...term_n)$
- $a = \neg r(term_1, ...term_n)$ updates the interpretation of $\mathcal{M}$ by removing an existing fact $r(term_1, ...term_n)$

Atomic formulas in dynamic logic are atomic first-order formulas of the form $\phi = r(t_1, ...t_n)$ where r is an n-ary relation symbol and $t_1, ...t_n$ are terms. An atomic formula $\phi = r(t_1, ...t_n)$ is true in a state *iff* the fact $r(t_1, ...t_n)$ exist in the domain of computation. Dynamic logic programs α are constructed using sequential composition ($\alpha_1 ; \alpha_2$), nondeterministic choice ($\alpha_1 \cup \alpha_2$), iteration (α_1^*) and tests ($\varphi?$) based on the grammar in [3]. Tests $\varphi?$, are special atomic programs whose execution terminates in the current state *iff* the test succeeds (is true), otherwise fails. Formulas φ are constructed using logical connectives (e.g., $\varphi_1 \rightarrow \varphi_2$), quantifiers (e.g., $\forall x \varphi$), and modal operators (e.g., $[\alpha]\varphi$) defined on programs. $[\alpha]\varphi$ defines many modalities, one modality for each program, each interpreted over the relation defined by the program α. $[\alpha]\varphi$ means that "if program α is started in the current state, then however (if at all) it terminates, in the final state, φ holds."

Once we have specified a program formally, we can use Hoare logic statements of the form $\{\varphi\}\alpha\{\psi\}$ for program verification which says that the program α is partially correct with respect to the pre-condition/post-condition specification φ, ψ. The pre-conditions/post-conditions specify that, if the program α is run in a state satisfying φ, then if and when it halts, it does so in a state satisfying ψ [5].

In order to use first-order dynamic logic in DPF, we need to establish that a typed graph can be converted to and from a first-order structure. The conversion is very straightforward and an example is shown in Figure 8 where a typed graph is converted to a first-order structure. During the conversion, for each node and edge from the type graph we get a unary predicate symbol. These unary predicates along with two binary predicates *src* and *trg* are added into the first-order vocabulary. The *src* and *trg* predicates are used to represent the source and target information of the edges of a typed graph.

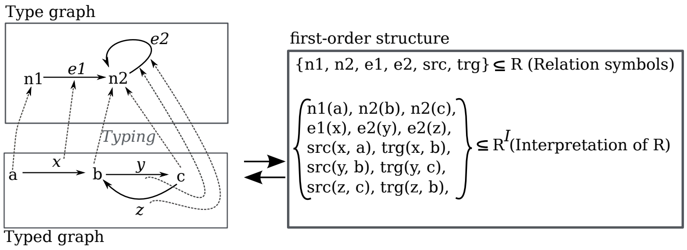


Figure 8: Conversion of a Typed graph to a first-order structure

With the proposed variant of first-order dynamic logic proposed, we can specify a program to manipulate a first-order structure. This feature along with the conversion mechanism enables us to specify a DL program that operates on DPF models. We use this feature to refine distributed applications. To illustrate this, we continue with the blood transfusion application and refine it with a DL program specification. Below the program specification for the process instance ':Check blood Type' from the workflow presented in Figure 5:

$$\begin{aligned}
\alpha &:= \alpha_1;\ \neg(\mathrm{z} = \text{``''})?\ \alpha_2;\ (\mathrm{z} = \text{``''})? \\
\alpha_1 &:= \mathrm{Patient(p);\ patientID(pid);\ src(pid, p);\ trg(pid, y)} \\
\alpha_2 &:= \mathrm{String(z);\ bloodType(bt);\ src(bt, p);\ trg(bt, z)}
\end{aligned}$$

The program uses the variables (e.g., y,z,p,bt,pid) from its input graph as shown in Figure 7 on the arc/edge from ':Get Patient Info' to ':Check blood Type'. The program α first executes program α_1 that creates new facts for patient instance and its relation with the patientID. The program α then checks if the bloodType is an empty string or not. If the test $\neg(z = \text{``''})$? succeeds (i.e., the blood type is not an empty string), the program α_2 is executed. The program α_2 creates new facts for blood type and its relation with the patient instance. The updated domain of computation is then used to update the instance of the Blood-App entity model as shown in Figure 9.

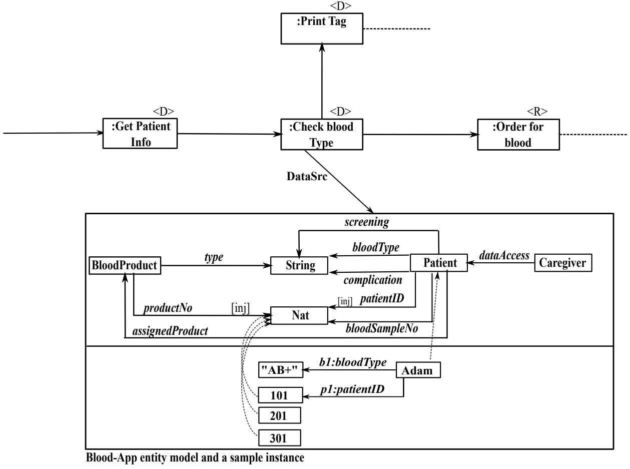


Figure 9: Updated state of the system after the execution of program α

6 Related and Future Work

There are several workflow and process modelling languages such as YAWL [13], BPMN [15], CWML [14], π-calculus [6] Tools have been developed to produce models with these modelling languages; some tools are also equipped with anlysis and verification techniques. MacCaull et al [16] developed a workflow management tool suite called NOVA Workflow where one can graphically design a workflow model using a visual editor and can write task specifications using the domain specific language $T_{\Box}$ [17]. Although these modelling languages have certain strength in modelling and analysis of a system they do not follow a metamodelling hierarchy and therefore the languages cannot be adapted. In this paper we provided a multi metamodelling approach which can be easily customized depending on various modelling requirements.

Diskin et al. proposed a megamodeling framework based on graphs and graph mappings, and operations over them [9]. They presented *model mapping* which is a structured set of links between models, specified a library of elementary building blocks, and presented how to combine them into a complex workflow. Diskin et al focused on a single aspect of a system with multiple views. Different views of the entity model in [9] would correspond to different views of the datasource model in our paper. The foundation of our multi metamodelling is based on DPF [1] and DP-logic which was initially developed by Diskin et al in the functorial semantics setting [10], [11]. In this paper we proposed an extension to the metamodelling approach for modelling distributed systems.

Rabbi et al [12] proposed a formalization for the co-ordination of multiple metamodels using a linguistic extension of the metamodelling hierarchy. The linguistic extension consists of co-ordinating edges together with constraints on those edges which allows the integration of metamodels. The proposed method can capture different aspects of a system and provides modelling notation to design an integrated model. In this paper we have used co-ordination between the process model and entity metamodels, and have mainly focused on modelling distributed systems. The use of typed graph for message passing communication, the proposed variant of first-order Dynamic Logic for writing process specification are new contributions of this paper. The proposed method provides different level of abstraction for modelling. In the future we plan to integrate the proposed approach into the DPF framework *http://dpf.hib.no* and perform evaluation of its applicability. We also plan to incorporating various analysis techniques to verify different aspects of models such as state space analysis for process models, consistency checking for models, and program verification for specifications written in dynamic logic.

References

[1] A. Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, Department of Informatics, University of Bergen, Norway, 2010.

[2] Y. Lamo, X. Wang, F. Mantz, W. MacCaull, and A. Rutle. Dpf workbench: A diagrammatic multi-layer domain specific (meta-)modelling environment. In R. Lee, editor, *Computer and Information Science 2012*, volume 429 of *Studies in Computational Intelligence*, pp. 37–52. Springer, 2012.

[3] D. Harel, T. David, and D. Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.

[4] A. Rutle, W. MacCaull, H. Wang, and Y. Lamo. A metamodelling approach to behavioural modelling, In *Proceedings of the Fourth Workshop on Behaviour Modelling - Foundations and Applications*, ser. BM-FA '12. New York, USA: ACM, 2012, pp. 5:1–10.

[5] M. Huth, and M. Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press. New York, USA, 2004.

[6] J. A. Bergstra. *Handbook of Process Algebra*. In *Elsevier Science Inc.* New York, USA, 2001.

[7] C. A. R. Hoare. *Process Algebra: A Unifying Approach*. In *Communicating Sequential Processes. The First 25 Years*. LNCS, vol. 3525, pp. 36–60. Springer, Heidelberg (2005).

[8] C. A. R. Hoare. *Communicating Sequential Processes*. In *Prentice-Hall* 1985.

[9] Z. Diskin, S. Kokaly, and T.Maibaum, . *Mapping-aware megamodeling: Design patterns and laws.* In *SLE; 2013* vol. 8225 of LNCS. Springer; 2013, p. 322–343.

[10] Z. Diskin, and B. Kadish. *Variable set semantics for keyed generalized sketches: formal semantics for object identity and abstract syntax for conceptual modeling.* Data Knowl. Eng. 47(1), 1–59 (2003)

[11] Z. Diskin, B. Kadish, F. Piessens, and M. Johnson. *Universal arrow foundations for visual modeling.* In *M. Anderson, P. Cheng, V. Haarslev. (eds.) Diagrams 2000.* LNCS (LNAI), vol. 1889, pp. 345–360. Springer, Heidelberg (2000)

[12] F. Rabbi, Y. Lamo, and W. MacCaull. *Co-ordination of Multiple Metamodels, with Application to Healthcare Systems.* In *EUSPN/ICTH 2014:* pp. 473-480

[13] W. van der Aalst, and A. H. M. T. Hofstede. *YAWL: Yet another workflow language.* In *Information Systems,* pp. 30:245–275, 2003.

[14] F. Rabbi, H. Wang, and W. MacCaull. *Compensable workflow nets.* In *In ICFEM 2010,* vol. 6447 of LNCS. Springer; 2010, p. 122–137.

[15] S. White. *Introduction to BPMN.* Technical report, 2004.

[16] W. MacCaull, F. Rabbi. *NOVA Workflow: A Workflow Management Tool Targeting Health Services Delivery.* In *FHIES 2011.* LNCS, vol.7151, pp. 75–92, Springer (2012).

[17] F. Rabbi, and W. MacCaull. *$T_{\Box}$: A Domain Specific Language for Rapid Workflow Development.* In *MODELS 2012.* LNCS, vol. 7590, pp. 36–52, Springer (2012).

[18] C. Atkinson, and T. Kühne. *Model-Driven Development: A Metamodeling Foundation.* In *IEEE Software,* 20(5):36–41, 2003.