Co-ordination of Multiple Metamodels, with Application to Healthcare Systems

Fazle Rabbi∗, a, b, Yngve Lamo a, Wendy MacCaull c

aBergen University College, Bergen, Norway
bUniversity Of Oslo, Oslo, Norway
cSt. Francis Xavier University, Antigonish, Canada

Abstract

Due to their complexity and the plethora of requirements placed upon them, healthcare systems so far have not been adequately modeled for the purpose of software development. As a result, the healthcare software suffers from high development costs and lack of flexibility. Model driven software engineering (MDSE) is an emerging methodology for software development, targeting productivity, flexibility and reliability of systems; metamodelling is at the core of most MDSE approaches. In previous work, we proposed a multi metamodelling approach that captures the complexity of these systems by using a metamodelling hierarchy, built from individually defined metamodels, each capturing different aspects of a healthcare domain, namely, user access modelling, health process modelling, process monitoring, user interface modelling and modelling of the data sources. Here, we formalize the co-ordination among metamodels, using a linguistic extension of the metamodelling hierarchy. This linguistic extension is an added metalevel which models the integration of two or more different aspects of the system. We focus on two features essential to the co-ordination of healthcare metamodels, namely the integration of process and data, modelling data-aware processes, and the integration of process and user, modelling both user access as well as inheritance of user access to tasks.

Key words: multi metamodelling; co-ordination among metamodels; linguistic extension; model driven software engineering; healthcare information systems

1. Introduction

Rising costs, ageing populations and increased expectations are making the current healthcare systems in the developed world unsustainable. Information technology has the potential to support healthcare but its application to support the continuum of care has not nearly reached its full potential. Barriers include the proliferation of systems even within one hospital, often which do not support interoperability; the fact that systems must be highly customized to adequately serve local situations (usually an expensive and error prone process); the fact that the systems are constantly evolving due to new medications, and changing protocols and management strategies; the fact that software engineering itself for such safety critical systems as healthcare needs new strategies to ensure that systems behave

* Corresponding author. Email address: Fazle.Rabbi@hib.no
Model driven software engineering (MDSE) is an emerging and promising methodology for software development, targeting challenges in software engineering relating to productivity, flexibility and reliability of systems. The construction of various kinds of models (e.g., blueprints, mockups etc.) is a well-known approach in the more traditional engineering fields; these models are used as artifacts to enable engineers to describe designs and validate whether a proposed design has desired qualitative and quantitative properties. We propose to use MDSE in an analogous manner for the development of reliable and robust software systems supporting the diverse, evolving and often safety critical requirements of healthcare. We note that many complex systems are characterized by the features discussed above and as such, the discussion here may be generalized to numerous processes involving complex information systems, such as transportation systems, communication systems, etc.

Many different MDSE technologies automatically generate code from models\textsuperscript{12}; these technologies are particularly suited for specifying the structural aspects of software systems generally, whereas the actual behavior is programmed manually. Some technologies for behavioural modelling in MDSE exist\textsuperscript{3,4}, but current approaches are often at a low level of abstraction and lack domain concepts for specifying behavior\textsuperscript{5}.

A collaborative group of researchers in Norway and Canada have been working on various issues relating to these problems. We describe a formal approach to workflow modelling in\textsuperscript{6}, based on the Diagram Predicate Framework (DPF)\textsuperscript{7} which provides a formalism of (meta) modelling and model transformations based on category theory and graph transformations\textsuperscript{8,9,10}. The dynamic semantics of models were described by a transition system where the states are instances and transitions are applications of coupled transformation rules. Our long term goal is to promote automatic generation of complex software systems from models, which can be easily adapted as the need arises; the formal approach permits a variety of formal validation techniques which go a long way toward ensuring correctness.

In his thesis\textsuperscript{11}, Baarah proposed an application framework for care process monitoring that collects and integrates events from event sources, maintains the individual and aggregate state of the care process and populates a metrics datamart to support performance reporting. He presented a UML-style metamodel for the care process monitoring application that had 3 main components: a process model, a performance model and an enterprise model. This work motivated us to take a more modular approach to our metamodelling, in order to more adequately model the various components involved in a healthcare process. In\textsuperscript{12} we proposed that the separation of concerns via the use of multimodelling was an appropriate methodology for designing healthcare information systems. The use of multi metamodels for modelling different aspects of a system facilitates abstraction and require less coupling among the models; this modularity gives flexibility as it permits the independent remodelling of parts of the system. It also reduces development and maintenance costs. The paper outlined several metamodels, each of which comes equipped with its own domain specific language which supports one aspect of the system and outlined where some of the links were needed for co-ordination. We focused on user aspects and monitoring, and the need for flexibility in design of user interfaces to suit the needs of different participants in the healthcare process. Users may interact with parts of the workflows, and parts of the datasource and receive and acknowledge alerts from the monitoring system.

In this paper, we begin the formalization of the co-ordination among the metamodels necessary to integrate two or more aspects during different phases of modelling. We use a linguistic extension of the metamodelling hierarchy, in the form of a metalevel which sits at the top of the multi metamodelling hierarchy and which provides co-ordinating edges which link different metamodels together with constraints relating these co-ordinating edges to edges in the existing metamodels. This metamodelling formalism can be used to automatically generate program code from models to implement software systems. In section 2, we briefly outline the DPF modelling approach, and use it to describe two metamodels, namely, the process metamodel and the datasource metamodel. In section 3, we describe the enrichment of DPF by defining the linguistic extension of the metamodelling hierarchy which permits the integration of metamodels. We discuss the co-ordination of data, by the integration of the process and datasource metamodels via a co-ordinating edge between them, subject to a commute constraint, ensuring that the data is assigned to the
appropriate patient. We also discuss the co-ordination of user privileges, by integrating the process and user access metamodels by means of a co-ordinating edge between them which models user access and also models inheritance of user access to workflow tasks. In that section, we also give an overview of five aspects of healthcare information systems and some required co-ordinating edges between them. In section 4, we briefly discuss related and future work. Section 5 concludes the paper.

2. Metamodelling in DPF

A traditional modelling hierarchy consists of 4 levels, meta-metamodels, metamodels, models and instances of models13. A meta-metamodel represents a modelling formalism, modelling languages are represented by metamodels, software systems are represented by models and possible instances of a software system are represented by instances of models. The metamodel describes the syntax of the modelling language, i.e., it describes the types and relations between the types. Each model must conform to the language’s metamodel, i.e., it must respect the typing and other constraints of the language. Finally, instances must conform to models. In contrast to traditional hierarchies, DPF7 has a potentially unbounded number of metalevels14. A model is represented by a diagrammatic specification \( S = ( S, C_S : \Sigma ) \) consisting of an underlying graph \( S \) together with a set of atomic constraints \( C_S \) specified by a predicate signature \( \Sigma \). A predicate signature consists of a collection of predicates, each having a name and an arity (or shape graph). A predicate (name) imposes a constraint on a portion of the graph \( S \), of the associated arity.

DPF provides a formalisation of multi level metamodelling by defining the conformance relation between models at adjacent levels of a metamodelling hierarchy. There are two kinds of conformance: typed by and satisfaction of constraints.

In6 we detailed the DERF modeling hierarchy developed to represent processes. Traditional DPF models were used to represent structural (static) aspects of a software system. The DERF modeling hierarchy extended modelling to include behavioural aspects (dynamic semantics); behaviour was defined by a coupled transition system. As an illustration we briefly discuss the process formalizing the Clinical Practice Guideline for the Management of Hypertension15. In the management of hypertension, an initial blood pressure (BP) measurement is made; if the BP is normal, the patient is “Safe”, i.e., the process stops. If the BP is high, the patient is scheduled for Visit 1, the first visit in the treatment process. An overview of the model of the treatment process is found in the lower portion of Fig. 1, the reader will note the [XOR SPLIT] predicate indicating that exactly one of the paths is followed at the Initial BP Measure task. This model uses predicates [XOR SPLIT], [AND SPLIT], [OR SPLIT], etc., to describes the control flow of the process model. The metamodel level provides the abstract modelling language for workflow, and consists of two types, Task, and Flow; the model level details a specific workflow; the instance level gives instances of specific enactments of a workflow. Fig. 1 shows two meta level specifications \( S_W \) and \( S_W \) for this process (the instance level is not shown). Arrows from the bottom layer to the top layer indicate the typing constraints. In the examples of the DPF modelling hierarchies given here, predicates at each level are described by their name and a “shape graph”, for visualization. The metamodel specification \( S_W \) of Fig. 1 is similar to a workflow metamodel specification presented in6. Space does not permit us to discuss this modelling hierarchy and its dynamic behaviour further; the reader can refer to6,12 for more details.

Fig. 1 provides a process flow modelling hierarchy for the hypertension management system but it does not specify the data model of the system. Let us consider the following data requirements from the healthcare domain which we use to describe the datasource modelling hierarchy: (We remark that these requirements are overly simplified from real life applications, but suffice to illustrate the features we wish to discuss here.)

**Requirement 1** An employee (e.g., nurse, doctor) must work for a department.

**Requirement 2** A department may have zero or more employees.

**Requirement 3** A ward must be under a department.

**Requirement 4** An employee who is involved in a ward, must work in the controlling department.

**Requirement 5** A patient’s systolic and diastolic blood pressure records are numbers.
Table 1. Predicate constraints of the signature $\Sigma_D^{\text{SD}2}$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\alpha_{p}^{SD2}$</th>
<th>Proposed Vis.</th>
<th>Semantic Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(multi($n,m$))</td>
<td>$f \leftarrow [n,m]$</td>
<td>$X$</td>
<td>$\forall x \in X : m \leq</td>
</tr>
<tr>
<td>(irreflexive)</td>
<td>$f \leftarrow [1]$</td>
<td>$X$</td>
<td>$\forall x \in X : x \notin f(x)$</td>
</tr>
<tr>
<td>(surjective)</td>
<td>$f \leftarrow [1]$</td>
<td>$X$</td>
<td>$\forall y \in Y, \exists x \in X, f(x) = y$</td>
</tr>
<tr>
<td>(inverse)</td>
<td>$f \leftarrow [1]$</td>
<td>$X$</td>
<td>$\forall x \in X, \forall y \in Y : y \in f(x) \iff x \in g(y)$</td>
</tr>
<tr>
<td>(image-inclusion)</td>
<td>$f \leftarrow [1]$</td>
<td>$X$</td>
<td>$\forall x \in X : f(x) \subseteq g(x)$</td>
</tr>
<tr>
<td>(composition)</td>
<td>$f \leftarrow [1]$</td>
<td>$X$</td>
<td>$\forall x \in X : h(x) =</td>
</tr>
</tbody>
</table>

In Fig. 2, we outline two meta level specifications, $\Sigma_{SD1}$ and $\Sigma_{SD2}$, of the datasource metamodeling hierarchy that we developed from the above mentioned requirements. In this specification we used the signature $\Sigma_D^{SD2} = (\Sigma_{SD2}^{\text{SD}2}, \alpha_{SD2}^{\text{SD}2})$ ($\Sigma_D^{SD2}$ indicates the signature at meta–level 2) from Table 1 to define the set, $C^\Sigma_{SD1}$, of atomic constraints where the first column of Table 1 shows the names of the predicates; the second, third and fourth columns show the arities of
predicates, possible visualizations, and semantic interpretations, respectively (Fig. 2 does not show all of $\Sigma^J_0$). The predicate constraints are added into the specification by a set of atomic constraints; this is depicted in the specification by using the proposed visualization. Requirement 1 is encoded in $\mathfrak{S}_{D_1}$ by the [surjective] predicate constraint over the morphism $\text{depEmps}$; the fourth requirement “an employee who is involved in a ward, must work in the controlling department” is encoded in $\mathfrak{S}_{D_1}$ by two predicate constraints [composition] and [image – inclusion] over morphisms $\text{depEmps}$, $\text{wardDeps}$, $\text{wardEmps}$ and $\text{wardEmps}'$ where $\text{wardEmps} := \text{wardDeps}; \text{depEmps}$ (the composition of morphisms $\text{wardDeps}$ and $\text{depEmps}$). The atomic constraint for [composition] predicate in $\mathfrak{S}_{D_1}$ indicates a morphism $z \xrightarrow{\text{wardEmps}} x$ exists iff morphisms $z \xrightarrow{\text{wardDeps}} y$ and $y \xrightarrow{\text{depEmps}} x$ exist, and the atomic constraint for [image – inclusion] predicate in $\mathfrak{S}_{D_1}$ indicates that for any Employee $x$ working under a Ward $z$ (i.e., $z \xrightarrow{\text{wardEmps}} x$) there is a morphism $z \xrightarrow{\text{wardEmps}'}$ to $x$. In this situation, $\text{wardEmps}$ morphisms are explicitly defined while $\text{wardEmps}'$ morphisms are derived from the existance of $\text{wardDeps}$ and $\text{depEmps}$.

3. Co-ordination of metamodells

In this section we describe a linguistic extension to the DPF metamodelling hierarchy; this extension is an added meta-level which models the integration of two or more different aspects of the system. The formalization of the linguistic extension, that is, the top layer of the hierarchy with two different types of edges, is inspired by the notion of E-Graph$^{16}$. To illustrate co-ordination, we present two features essential to the co-ordination of healthcare metamodelling. Firstly, the integration of process and data, modelling data-aware processes, and the integration of process and user, modelling user access to tasks in a process as well as inheritance of user access.

![Fig. 3. Example integration of a process metamodel and a datasource metamodel](image)

3.1. Co-ordination of a process model and a datasource model

The process model shown in the previous section is not data-aware. In order to develop a data-aware process model we need to co-ordinate the process model with a datasource model. In Fig. 3 we present the integrated modelling hierarchy where $\mathfrak{S}_1, \mathfrak{S}_2, \mathfrak{S}_3$ are specifications of meta-levels representing the co-ordination of the process metamodel and the datasource metamodel. Our linguistic extension is based on ideas from$^{17}$. The formalization of the specification $\mathfrak{S}_3$ (similar to that in$^{17}$) uses, in addition an “ordinary” edge (called Edge), a so-called “Co-ordinating Edge”, subject to constraints. The specification $\mathfrak{S}_3$ contains a signature $\Sigma^J_3$ with a predicate called $\text{mult}(n, m)$. The semantics of the predicates containing “Co-ordinating Edge” is defined in a fibred manner as in$^{7}$. That is, the semantics of a predicate $p$ is given by the set of its instances. $\mathfrak{S}_3$ consists of specification $\mathfrak{S}_{W_2}$ and $\mathfrak{S}_{D_1}$ and a signature $\Sigma^J_3$ with one predicate [commute]. Moreover it has a co-ordinating edge, DataReference from Task in $W_2$ (underlying shape graph
3.2. Co-ordination of a process model and a user metamodel

The access policy of a task may be specified by an integrated metamodelling hierarchy shown in Fig. 4. The Copy-acc edge of the User Access specification \( \mathcal{S}_U \) copies access from one user to another. The specification \( \mathcal{S}_J \) has a co-ordinating edge, TaskAccess, which co-ordinates the user access to a task. Both the TaskAccess and Copy-acc arrows are labelled with the composition constraint, \([comp]\), which ensures the inheritance of task access between users. The arrow \( h \) at the bottom level is a co-ordinating edge and copy-acc arrow \( c \) from Doctor to Nurse indicates that the Doctor inherits the access that the Nurse has to the tasks. The co-ordinating edge \( s \) indicates that the Doctor has access to the task Measure.

Table 2 illustrates an example where Doctor Bryan is in relation with Patient 1 and therefore has access to all Patient 1’s data. Doctor Bryan can see only Patient 2’s BP-related information since he copied (inherited) the access of Nurse Jessica who has access only to Patient 2’s BP information. Other fine-grained access policies may be defined; e.g., only allowing a doctor to copy (inherit) access regarding a patient from a nurse, if the patient the nurse is treating is one of the doctor’s patients.
3.3. Co-ordination of different aspects

In Fig. 5 we show 5 aspects of Health Information Systems, namely, a Monitor aspect, a User Interface View aspect as well as the Process, Datasource and User Access aspects discussed here. See12 for details on the metamodels for the two aspects not discussed here as well as an example of the co-ordination among 3 metamodelling hierarchies (User Access, Process and Monitor) required for the Monitor to give alerts. In Fig. 5 we show a number of co-ordinating edges: Trigger, Sends, Displays, Alert UI, View Access, as well as other examples of Data Access, which are required to co-ordinate the 5 aspects displayed.

4. Related and Future Work

Many workflow languages such as YAWL18, ADEPT219 focus on a single aspect of a system, usually the control flow. YAWL is data-aware workflow language which supports XML-based models for data definition and manipulation based on XML Schema. These languages are not flexible enough to customize to a different syntax or semantics. In this article we enhanced the DPF framework using an integrated metamodelling technique which can be customized according to different needs. The approach makes the framework modular and flexible yet structured as an integrated model specification has to satisfy typing and conformance to its metamodel specification.

An OCL-based domain specific language, MoScript, was presented in20 for model-based task and workflow automation. The language is based on the metadata contained in a Megamodel that allows querying a Megamodel for the required modelling artefacts and tools. It supports model manipulations, such as loading and storing modelling artefacts, invocation of a model transformation engine, etc. MoScript is able to perform some validation at run-time such as to check if a model transformation is applied to a model that conforms to the right meta-model. Whereas in our work, we enhanced the software modelling with the co-ordination of multiple metamodels.

Diskin et al. proposed a megamodeling framework based on graphs and graph mappings, and operations over them21. 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. In that paper, the authors focused on a single aspect of a system with multiple views. Different views of the entity model in21 would correspond to different views of the datasource model in our paper. They presented a query mechanism and, correspondingly, q-mappings (Kleisli morphisms), essential for megamodeling such views. In this paper, we focused on several aspects of a system (datasource aspect, monitor aspect, etc.) and presented a metamodelling approach to integrate them by co-ordination. Our approach can be further improved by the use of a query mechanism which should incorporate more expressiveness in the DPF metamodelling language. In future we also plan to metamodel other aspects of healthcare information systems (eg., Scheduling, Resources, and Service Orientation) and co-ordinate them. This may require an extension of the metamodelling language with time22, and compensation23 and co-ordinating them; including such features as time and compensation are essential for the enactment of real life healthcare workflows. The implementation of the editor requires abstraction and projection facilities in order to accommodate many models in the display devices.
5. Conclusion

Separating different concerns (aspects) of a system into several metamodels gives us flexibility, allowing us to modify one aspect of a system described by a particular metamodel without affecting other metamodels. However, to realistically capture the requirements of complex systems such as healthcare systems we need to integrate the metamodels. In this paper, we proposed a linguistic extension to the DPF metamodeling hierarchy, consisting of co-ordinating edges together with constraints on these edges, this allows us to integrate metamodels, thereby modelling features that cannot be modelled by the multi metamodels in isolation. We were able to capture data-aware processes through the use of co-ordinating edges together with the constraint represented by the commute predicate, and the inheritance of user access using a co-ordinating edge together with the constraint represented by the composition predicate. We believe that multi metamodeling with linguistic extensions as described here, together with MDSE principles in general, can be used as the main methodology in the development process of software for care processes.

References