Generating Information Systems applications: a research proposition

Betsy Pepels[1,2] and Rinus Plasmeijer[1]

[1] Software Technology Department
Radboud University Nijmegen,
The Netherlands

[2] Informatics and Communication Academy,
HAN University of Applied Science,
The Netherlands

{betsy, rinus}@cs.ru.nl

Abstract

More and more enterprises are becoming more and more dependent on their computerized Information Systems (IS's). Yet the development of IS's of high quality and their adaptation to changes is a major challenge. With current IS development methods it is hardly possible to tackle the complexity nowadays IS's bring about. Functional Languages are a promising means to model IS's; based on such a model at least a substantial part of an IS application could be generated automatically.
In this position paper, we set out a research agenda for designing an IS modeling language based on features of Functional Languages, and for realizing fully automated IS application generation.

1 INTRODUCTION

In the information age, enterprises use computerized Information Systems (IS's) to manipulate their vital data. Over time, the complexity of these IS's has grown tremendously. In the early days of the Computer Era, IS's just stored plain data like addresses and insurance policies of clients. Modern IS's record and manage business processes. For instance, IS's of a car production plant register (amongst others) the options of the car for an individual costumer, and control the schedule of the building of the cars, the just-in-time deliverance of parts, the transportation of the parts to the right place in the plant, and the actual building of the cars.

More and more enterprises are becoming more and more dependent on their IS's. The *quality* of their IS's determines to a large extent the vitality of enterprises: the more IS's serve the information needs and the more IS's are aligned to the business processes of an enterprise, the more competitive it will be. Yet the development of IS's of high quality and their adaptation to changes is a major challenge. With current IS development methods it is hardly possible to design the complex systems needed; the implementation of IS's has mainly to be done by hand and is hence very expensive.

Functional Languages are useful for developing IS's. They offer high expressive power and a high degree of reusability, thus making design of complex systems easier. They can be used as an executable specification language; with modern techniques like generic programming, code can be generated from the types being specified. In this way, the amount of hand coding can be significantly reduced. This position paper

- analyzes the background of the problems with IS Development (section 2)
- introduces a sound approach for IS design (the family of Fact Oriented Modeling languages) and explores how Functional Languages could be used to enhance the expressive power of these languages (section 3)
- defines a research agenda for generating IS applications (section 4)

Research for generating IS's has many relationships with other research, which we describe in section 5. Section 6 concludes.

2 INFORMATION SYSTEMS DEVELOPMENT

Information Systems Development differs in many ways from academic programming languages research and its applications. To fully appreciate the peculiarities of getting into research for new ways of ISD and IS application generation, we first elucidate the current process and the culture of IS Development (ISD).

2.1 Idealized Information Systems Development

In theory, the development of an Information System is a relatively well defined software engineering process [1] [2] [3]. All kind of methods and Computer Added Software Engineering (CASE) tools are extensively available and are being used to support the development process.

IS's are usually developed by a multidisciplinary team, each member contributing according to his or her specialty. Ideally such a team walks through a number of phases. Depending on the specific method, the number of phases, their exact activities and their names differ. There is however a general pattern: first it is specified *what* what the system should do and then *how* this is accomplished, before the system is actually *constructed* (coded). For this summary, we adopt a subdivision into three main phases.

First the system requirements are settled: the *specification* phase. These are laid down in a document extensively describing issues as data to be stored, functionality to be fulfilled, layout of screens, security measures and performance demands. For serious systems, this document might be several centimeters thick.

Then the *design* of the IS is carried out. It is commonly agreed that proper IS development has its basis in *data* or *information modeling*: the analysis of what data should be stored and its interrelationships. Data modeling requires a good understanding of the Universe of Discourse (UoD). A data modeling *method* should provide a means of specifying this understanding in a clear, unambiguous way. The result of this analysis is the *Information* or *Data Model.* Mainstream methods for data modeling are Entity Relationship Modeling (ER or ERM) [4] and Unified Modeling Language (UML) [5].

Next the IS is actually *built*. This involves mapping the data model to a relational scheme and coding all kind of queries (usually in SQL); and furthermore the coding of the so called business logic and the user interface (usually in an imperative programming language like Visual Basic or Java). The system under construction is extensively tested and finally delivered to the users.

2.2 Not-so-perfect Information Systems Development

Most often, ISD projects are carried out in what is euphemistically called a more pragmatic way [6].

Background and culture The vast majority of IS's serve the need of enterprises for recording their data. These IS's register for instance data about clients, suppliers, patients or students. Besides registering data, the most important reason of existence of IS's is to adequately support the *business processes* of the enterprise, like *placing an order* or *registering a complaint*. Moreover, building new information systems often involves major organizational changes, requiring serious commitment of managers. As a "natural" consequence, the culture of IS development traditionally is heavily influenced by the business operations framework.

This results in an emphasis on project management (with lots of reports and fancy PowerPoint presentations) and on the possible business opportunities (phrased in *management speak*), a relative neglect of the software engineering perspective (also seen in the salary of the programmers) and a complete unawareness of the existence of a formal approach. It is amazing how many times words like *system*, *process*, *abstract*, *formal* and *model* appear in the paper pile produced in an ISD project - formalizations however being completely absent.

Approach Since IS development is not regarded as an *engineering* activity, the staff responsible for design and implementation is often not trained accordingly. Programmers rarely have a formal background in Computer Science or whatsoever, but took for instance only some short product courses. As a consequence many best software engineering practices are disregarded. Often the system requirements are defined only globally, or not at all. Often no (formalized or conceptual) design is made. It is immediately started with the development of the relational scheme itself. Neglecting technical challenges in the early phases is also politically pleasing: one can present the project as running fine and meeting deadlines (although only because the real difficulties didn't come to light yet).

In the later phase(s) of the project - when the actual system is to be built - the problems return like a boomerang. At the end, programmers are producing tons of code whilst not knowing what the system precisely should do neither whether this is the right way to accomplish it.

Discussion For an ISD project, there are convincing reasons for heavy involvement of business oriented managers. The introduction of new IS's almost always induces (major) organizational changes, or even the other way around: because of organizational changes, new or adapted IS's are needed. ISD projects are often large complex undertakings. Serious involvement of capable project managers is crucial, as is commitment of the organizations' line management. The engineering perspective should however not be neglected and well educated technical staff should be responsible for the design and construction of the system.

2.3 Problems with data modeling methods

Even if an ISD team uses a proper approach and applies an appropriate data modeling method, there are still a lot of problems.

Mainstream modeling methods (ERM being the most important) are developed as a practical solution to a practical problem: for modeling administrative information (as found in many organizations) that are to be stored in relational databases. The modeling methods focus on arriving at a scheme for efficient storage and retrieval of data. As an undesired side effect, the methods are less suited for modeling complicated UoD's with complex data interrelationships.

The family of entity-relationship modeling methods considers the UoD to be described in the form of *objects* or *entities* with *attributes*, having *binary relations* with each other.

For instance, an employee working for an employer is modeled as two entities (*Employee* and *Employer*) having the relationship *is employed by* (or, reversely, *employs*). A possible attribute of Employee might be *Address*.

Entity-relationship based modeling methods suffer from fundamental conceptual problems. Our first criticism is that this approach hardly doesn't offer abstraction mechanisms compared with designing the relational scheme directly. An entity is just a relational table; a relation is directly to be translated to a foreign key. Another shortcoming is that ternary and higher relationships have to be simulated. For instance, when we want to model the *Hire Date* of the *Employee*, this is not possible in a direct manner. *Hire Date* is neither an attribute of the Employee nor of the Employer. In this case, the relationship has to be promoted to an object (for instance *Employment*) and then an attribute like *Start Date* is to be attached to it.

More general, our criticism is that the three basic concepts *object* - *attribute* - *relationship* often cannot clearly be distinguished from each other. For example, *Postal Address* might be an attribute of *Employee*, but might be also an entity of its own, with which *Employee* has the relation *has*. Or we might have an entity *Delivery Point* (for surface mail) with which *Employee* has the relation *uses as Postal Address*. Such a vague conceptual basis for a modeling method is very questionable.

Many abstraction mechanisms known in Functional Programming Languages are completely absent in ERM methods. There is no way to define complex data structures, like a *Tree*. Therefore it is also not possible to define a general polymorphic *Tree* and to successively define tree-like structures as instances hereof. Each tree-like structure in each scheme is to be simulated anew by (binary) father-son relationships, with rather complex queries for instance preventing the structure thus defined becoming cyclic.

2.4 Summary

The problems with data modeling methods can be generalized by regarding them from the perspective of programming languages. The most important ones are:

Lack of expressiveness Existing modeling techniques lack the capability for defining complex data types as found in modern programming languages (like algebraic, recursive or polymorphic data types); only the use of simple structures is supported. Complex structures cannot be expressed with appropriate types. As a consequence, the corresponding models have to resort to a mapping of the complex concept onto an enormous amount of fine-grained entities and attributes.

Lack of abstraction mechanisms: limited possibilities for reuse Some information is very similar to other information, for instance the personnel administration of one plant may have a lot in common with the administration of another. To reuse code that was developed for one system, one needs facilities to abstract from specific information. This kind of facilities is not available in current methods.

Lack of formality It is often unclear what the formal meaning is of the model made. ERM and comparable methods just aren't developed with formal semantics in mind. There is only some common agreement about how diagrams are to be sketched and about their approximate meaning. A formal semantics theoretically could be assigned to these models, but this wouldn't be very useful, since the methods are not used in a formal context.

Lack of completeness The emphasis of data modeling methods is on the development of the conceptual structure of the IS: the static part. They hardly address the specification of the behavior of the system. Furthermore, issues like end user interaction and the GUI aren't part of the method.

When it comes to actually building the IS, we observe additional problems:

Implementation is tedious and error prone Implementing IS's is repetitive work. After the relational scheme is set up and the user interaction is defined, it consists of mapping the data manipulation allowed to the end user to updates to the RDBMS. Little changes to either of these gives rise to to a considerable amount of coding work. Due to the large dimensions of IS's, it is hard to view the whole picture. This easily leads to errors.

Some of the problems described can be alleviated to some extent by using Object Oriented methods and languages, for instance UML and Java, allowing more powerful modeling concepts like classes, inheritance and template classes. UML also addresses modeling issues other than data modeling. This approach however causes problems to reappear in the implementation phase, in which the objects have to be mapped onto the relational data base. This is known as the object-relational impedance mismatch [7]. Furthermore, UML still is an entity/attribute/relationship based method with the peculiarities described earlier.

3 FUNCTIONAL PROGRAMMING AND GENERATING IS'S

There is no straightforward answer to the problems sketched above. Nevertheless, the current development process can be improved a lot. Clearly modelers could do better using a more expressive modeling language than current ones. There even migth be a better option. Given a suitable specification in a well defined information modeling language, it might be possible to generate the whole IS application fully automatically. This is were Functional Programming Languages (FPL's) and Generic Programming Languages come in. FPL's are well-known for their expressive power. Generic Programming techniques are very suited for generating algorithms. In this section, we sketch how FPL's can be used to empower an existing modeling language (Fact Oriented Modeling).

3.1 Starting point: Fact Oriented Modeling

There exists a family of related modeling approaches, the Fact Oriented Modeling (FOM) methods, like NIAM [8], Object Role Modeling (ORM) [9], FCO-IM [10] and PSM [11] and ORM2 [12]. All FOM methods are very close to each other and differ only in details. Several CASE tools supporting FOM are available, some commercial, e.g. Visio [13] and CaseTalk [14]. FOM has successfully been used in ISD projects where mainstream methods previously failed [15].

Through the years, much research on the FOM subject has been done, for instance to establish a formal underpinning of the semantics of FOM models, e.g. [16], to automatically generate GUI's [17] and on extensions of FOM languages (with concepts like *Set* and *Bag*) [18] [19].

3.2 Summary of Fact Oriented Modeling

Fact Oriented Modeling pictures the world in terms of *objects* (entities or values) that play *roles* (parts in relationships) in *facts*. The modeling process starts from examples, uses formalized natural language, as well as intuitive diagrams, and expresses the information in terms of simple or elementary *facts*, in contrast to ERM picturing the world as *entities*. FOM methods further ease the modeling process by a "cookbook" way of working.

For this summarizing overview of Fact Oriented Modeling, we use Object Role Modeling (ORM), the currently by far most used fact oriented modeling method. In this paper we can only address the main issues. For more detailed information, we refer to text books like [9]. As with most data modeling methods, FOM models are denoted as diagrams. Figure 1 gives an example ORM schema.

Modeling with FOM/ORM When making a FOM model, the modeler starts with phrasing elementary sentences about the Universe of Discourse, for example:

```
The course SE1 is taught in Fall 2005.
The course FP is taught in Spring 2005.
```

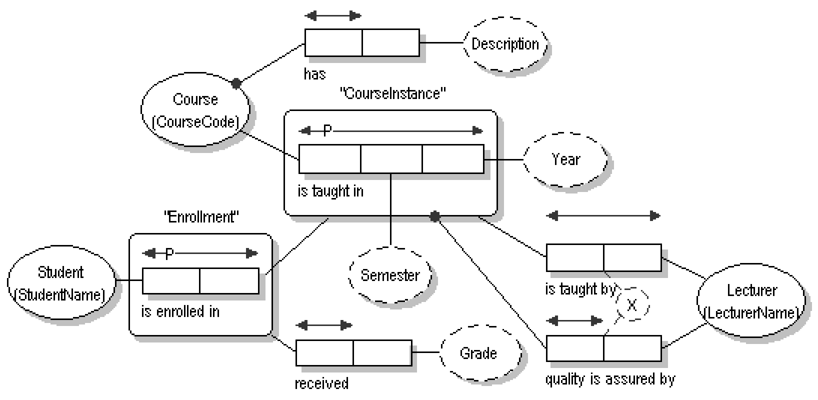

FIGURE 1. Example Object Role Model

These sentences are successively generalized into *fact types*:

`The course <CourseCode> is taught in <Semester> <Year>.`

In this sentence, *Course* in an entity type (referring to an object in the UoD) and *semester* and *year* are label types (just values in the UoD).

Entity types are identified by a definite description (for example, Course) and a *reference mode* (for example, CourseCode). Value types need no identification since they consist of a value only.

In the diagram, an entity type is depicted by an ellipse, and a label type by a dotted ellipse. A role is depicted as a box. Fact types are depicted as the composition of these boxes. A role is connected to its (unique) role player by a line segment. The number of its roles is called the *arity* of the fact type. FOM allows fact types of any arity. An n-ary fact type can also be seen as an n-ary relationship.

Object types can be *role players* in more than one fact type; for instance the entity *Course* plays two roles. Fact types can be role players their selves. Then the fact type has to be promoted to an entity type, called *objectification*. It is depicted by drawing an ellipse around it. For example, the fact type mentioned above is objectified to *CourseInstance*, playing a role in three other fact types.

Constraints Properties of the UoD are to be recorded as *constraints*. Constraints define the allowed populations in the FOM model. The constraints mostly used are the *uniqueness constraint* (UC) and the *mandatory role constraint* (MRC).

A uniqueness constraint concerns populations of a role or of role combinations. It declares that instances of that role (-combination) in the fact type population must be unique. A UC is depicted as an arrow tipped bar, and is placed over one or more roles in a fact type. For example, the UC over the first role of the fact type ”The course <CourseCode> has description <Description>.” declares that each course has at most one description. The UC over the both roles of ”The

<CourseInstance> is taught by <Lecturer>." states that course may have several lecturers and that lecturers may teach several courses.

A mandatory role constraint declares that every instance in the population of the roles object type *must* play that role. It is usually shown as a black dot. For example, the mandatory role constraint of the object type Course declares that each course must have a description.

There are many more predefined constraints. We here just mention the *pair exclusion constraint*, indicating that populations are mutually exclusive. An exclusion constraint is depicted as a circled X. For example, the pair exclusion constraint in the diagram states that the quality of a course may not be assured by a lecturer of that course.

Properties of populations that can't be caught by constraints predefined in FOM can be expressed as *general constraints*. These are usually phrased as plain text. An example is "A student is not allowed to enroll a course (s)he already passed.".

Discussion Fact Oriented Modeling differs in many ways from mainstream modeling methods like ERM and UML. The formalized natural language basis of FOM results in a more clear separation between model and implementation (relational tables) than with ERM. FOM has as basic structuring concepts only labels/objects and elementary facts, in contrast with the foggy concepts of ERM. Facts are basically n-ary relationships so that also non-binary relations can be modeled naturally. And last but not least, FOM is developed on a sound mathematical basis.

Yet many of the problems depicted in section 2.4 are present. This we address in the next sections.

3.3 Fact Oriented Modeling seen from the FPL research point of view

A FOM model has a close relationship with types as we know them in Functional Programming Languages: a FOM scheme is equivalent to a data type. Here we have the starting point of research that promises more powerful information modeling methods and tools generating IS applications fully automatically.

In this section, we explore how Functional Programming can contribute to information modeling and to IS application generation and on basis hereof arrive at research directions.

3.3.1 FOM models as types

The labels and objects in a FOM scheme we recognize as (basic or simple) types. For instance, a *Student* has type `StudentName` which could be a synonym type of `String`.

An entity type (a solid ellipse in the diagram) represents a population of the given type. For example, the entity *Student* denotes a `Population` of type `StudentName`. Such populations we could implement with e.g. (sorted) lists.

An n-ary fact type basically represents the `Population` of an n-ary tuple type. The type of the members of such a tuple type depends on the role player in the scheme. If the role player is a label type, the type of the corresponding member is just the type of the label type.

If the role player is an entity type, determining the corresponding type in the tuple type requires some attention. Entity types can play more than one role. For example, the same lecturer can play two roles (one as lecturer of a course and one as quality assurer). So we have to allow the *same* member of an entity population to play all roles of the entity type: members of entity type populations have to be *shared* in fact types. So in this case the member of the tuple type has as type a reference (pointer) to the type of the corresponding entity type. This interpretation of the scheme is conform the semantics of the FOM dialect FCO-IM [20].

The ORM scheme of figure 1 can be translated to the following type definitions, specified in the language Clean [21]. FPL's do not provide directly for sharing; we use the notation `RefTo` to specify a reference type.

```
:: CourseInstance :== (RefTo CourseCode , Year , Semester)
:: Enrollment :== (RefTo StudentName , RefTo CourseInstance)
:: FT1 :== (RefTo CourseCode , Description)
:: FT2 :== CourseInstance
:: FT3 :== (RefTo CourseInstance , RefTo LecturerName)
:: FT4 :== (RefTo CourseInstance , RefTo LecturerName)
:: FT5 :== Enrollment
:: FT6 :== (RefTo Enrollment , Grade)
```

Now that we have identified the object types of FOM being types of FPL's, an obvious step is to allow objects to be of other (more complex) types than basic types alone. In this way we obtain almost straightforwardly the expressive power of FPL types in FOM.

Research questions Here many research questions arise. For instance: can we allow any type indeed? What will be a suitable syntax of the modeling language thus designed? How about its semantics?

3.3.2 FOM models as executable specifications

Functional Programming researchers almost immediately recognize a FOM scheme as an executable specification of an IS application, though somewhat rudimentary. With the definition of the data structure (the FOM scheme), conceptually the IS application is determined to a large extend. The IS application should maintain valid populations of the scheme. As a first exploration, we built a prototype tool for generating IS applications [22] using this approach.

Research questions Again there are many. The FOM scheme specifies only the data structure of the IS application. Yet much more information is needed to generate an application, like the user interaction part. How do we do this? On basis of what techniques do we generate the application? How much information is possible to encode in types, such that generic algorithms can be used? We want to incrementally define Information Models and generate applications out of them. How can this appropriately be done?

3.3.3 Constraints

From the IS modeling perspective, constraints are used to express properties of the UoD. To maintain data integrity, constraints should not be violated. When implementing the corresponding IS application, only a very few of the standard constraints can be translated to constraints an RDBMS supports. For example, a NOT NULL constraint on a column might be used to implement a Mandatory Role Constraint. Most constraints are to be implemented by hand coding, like the pair exclusion constraint. General constraints have to be first interpreted by the developer and then this interpretation has to be coded too. All this has to be done for each constraint in each scheme anew.

From the Functional Programming perspective constraints can be considered as *predicates*: mathematical functions having a `Boolean` value as result. These predicates are part of the Information Model. In this approach, each predefined constraint of a FOM modeling language can be defined once (for instance comprised in a tool) and reused subsequently. General constraints are to be defined by the modeler as a predicate.

Formalizing constraint definitions in this way is valuable because there is no doubt any more about their meaning. But there is more. Constraints thus defined can be plugged directly into the generated IS application to maintain data integrity. No additional coding is necessary any more.

The strategy to accomplished this is as follows. Observe that constraints can be violated when changing the population of a model. So constraints should be checked upon change of populations they concern.

Example A pair exclusion constraint possibly can be violated when a new member is *inserted* or *updated* in one of the fact type populations on which the constraint is put (but not when members are *deleted*). We consider in this example the pair exclusion constraint for inserting a member into on of the populations. It can be defined as:

```
insExcl :: DB (DB FTId → [a]) FTId a → Bool
insExcl db retrF ftId new = not (isMember new (retrF db ftId))
```

When the appropriate call of `insExcl` delivers False, the insertion of the new member violates the constraint and hence should not be done.

In our prototype tool mentioned above, we use this approach for the recording of ORM schemes. In the tool constraints in the form of predicates are comprised expressing what schemes are valid. The tool checks these constraints after the developer has inputted a part of the ORM scheme, but before it is stored.

Research questions Constraints determine the changes allowed to the population of the model: the behavior of the system. Another way to define the behavior of the system is by directly defining the end user interaction. These two approaches are complementary yet they interfere with each other.What is a good approach to define the behavior of an IS?

4 RESEARCH PROPOSITION: GENERATING INFORMATION SYSTEMS

Based on the explorations described in the previous section we are now able to set out a research agenda for realizing complete IS applications on basis of an suitable specification (an Information Model). We currently are in the process of writing a proposal for such research. This research encompasses four overall objectives:

1. to develop a general IS application specification language, for defining the IS conceptual structure and the user interaction;
2. to develop a suitable method for incremental and evolutional definition of IS's;
3. to design and build an accompanying tool for incremental IS specification and generation, and for automated evolution;
4. to ensure applicability of the language and the tool in the IS development field.

By generating IS's, ISD improves in a number of ways:

Improvement in development time and costs The labor intensive mapping of the Information Model to the execution platform isn't necessary any more, and hence off-shoring. Also extensively establishing requirements beforehand becomes superfluous.

Improvement in system correctness and effectiveness Generated systems are less likely to contain errors. When a system is generated by ”one push on the button”, it can be defined and built incrementally whilst users giving immediate feedback; this improves the effectiveness and the acceptance of the system.

Improvement in maintaining business alignment Generating IS applications puts *system evolution* in reach. With evolution we mean the ability to adapt the Information Model of an existing application, and automatically generate corresponding transformations for the data from the old IS application to the new one.

Improvement in Information (Systems) Management When IS applications can completely be generated and be adapted on basis of their models solely, managing Information Systems becomes managing Information Models; the level in which is thought about IS's is raised from *code* to *model*.

Each of the research objectives mentioned above we elucidate.

4.1 IS application modeling language

The modeling language requires a very high expressive power in order to tackle awkward modeling problems. Furthermore, the language should be intuitive, having a modest learning curve, should be appealing, also to non-technical developers, and should somehow be useable in contacts with end-users and stakeholders of the IS application. These trade-offs require a careful language design. There are many research questions:

Syntax and semantics What kind of language constructs does the modeling language have? For instance what mechanisms for typing, abstraction and defining of functionality are suited? What is a minimal language needed to model information in a proper way? What will be the formal semantics of the modeling language?

Visual vs. textual language Most modeling languages have a strong visual component. Visual languages are very suited for overall design and for discussions in teams and with users. They are less suited for settling detailed specifications, for which text based languages are more adequate. How do we find a balance?

Application generation information How do we conveniently model information which is required for the generation of the IS, but is outside the data model itself (e.g. the interactive use of the information system, the GUI)?

4.2 Support for incremental and evolutional definition

Essential in our proposed approach is that modelers can incrementally define a series of Information Models, each successive model being closer to the model of the final Information System. Almost equally essential is that a running IS (an IS *in production*) automatically can be upgraded to a next version on basis of the new model, including transformations on data already present in the existing system. Generic algorithms are a promising technique [23].

Strategy How can incremental models be defined in a feasible way at all? Can then incremental definition adequately be supported by types and by generic algorithms? How is the developer adequately provided with feedback about changes to the model influence the generated system?

Generating data transformations How do we generate data transformations? Generic programming techniques are very promising, but can they tackle every transformation?

4.3 The tool

The tool will generate the IS application in conceptually two parts. The first part is the *Data Storage and Access* part. This part ensures that the information is stored in an appropriate way (*persistence*) and enabling efficient and consistent multi-user access. The second part is the *User Dialogue* part: a suitable (web based) graphical user interface. Both parts are to be executed in a dedicated application server being part of the tool too.

Techniques for generation How much can we generate using generic programming techniques? How do we accomplish this? What do we have to do conventional?

Persistence How can we efficiently store complicated data types, data and code in existing commercial information systems? Dynamics enable us to store any type of information in a persistent store [24] [25]. With what storage schemes do we obtain the efficiency needed? Can we define an interface layer that makes it possible abstract from concrete specific commercial RDBMS's?

User Interface Web based Graphical User Interfaces can be generated for any data type using generic programming techniques [26] [27]. A one-to-one mapping from database tables or arbitrary data types to interactive web pages is very probably not user-friendly enough. How is the end user allowed to navigate through the pages ensuring that the result is again a valid and sound Information System?

4.4 Applicability of results

The success of the research is determined to a major extend by the usability of the tool and of the generated IS's. Presumably, defining an academically oriented modeling language accompanied with a spartan tool will be perfectly feasible; but if modelers in the field aren't able to deploy the tool in their everyday work for whatever reason, it will be of little or no use. Another factor that should seriously be taken into account is the practice of developing systems without a proper engineering approach isn't changed by just offering a tool.

Ensure applicability A vital part of the research activities should be to continuously check the results with workers in the field. Is the language useable in everyday practice? Is the support offered by the tool adequate? Does the generated system comply with then demands of the end user? What is a good strategy for helping teams adopting a more engineering oriented approach?

5 RELATED RESEARCH

Functional Programming Languages research To our knowledge, in the FPL community limited research has been done previously in the area of IS application generation. Some FPL's have a library for expressing data base queries and operations in a type safe and declarative way. For instance, HaskellDB [28] compiles a relational algebra-like syntax into SQL, submits the operations to the database for processing, and returns the results as ordinary Haskell values. FPL's are also used in the related area of querying XML [29].

Model Driven Architecture The Model-Driven Architecture (MDA) [30] is a standard framework proposed by the Object Management Group (OMG) for enterprise application development. MDA separates specification of the operation of a system from the details of the way that system uses the capabilities of its platform. The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns. MDA is developed and described the for OO environments (UML and UML2, Java). MDA describes a collection of models and mappings between them. A distinction is made between a Platform

Independent Model (PIM) and a Platform Specific Model (PSM). Developers first design a PIM, then a PSM and a mapping from the PIM to the PSM, and finally implement the PSM on the execution platform. MDA can be used for (partly) automating the mappings, although it wasn't developed especially for this purpose.

Our approach is in some ways very similar to that of MDA. The developer specifies with the modeling language conceptually a Platform Independent Model (PIM); the tool provides for the automated mapping from the PIM to the execution platform. We go a step further than MDA and enforce the mapping of the PIM on the execution platform and the implementation itself to be automated.

Broader academic scope The idea of automatically mapping models is not new. For example, the ASF+SDF Meta-Environment is an interactive development environment for the automatic generation of interactive systems for manipulating programs, specifications, or other texts written in a formal language [31]. The same research group argues that grammars and all grammar-dependent software ("grammarware") should become a thorough and established engineering discipline. Our approach is a typical *grammarware scenario* as described in [32].

Application generation Generating in stead of hand coding software is a hot issue. The bulk of tool development for this purpose is done in industry, and only at relatively few universities. Many firms, mainly small or medium sized, offer tools for application generation. Generally, available tools have restricted purposes. Such a tool might be product or platform specific (for instance might be an add-on of a given Cobol environment), or generate only a part of the IS application (only the relational scheme, allow only to *drag and drop* a GUI), or generate parts of each tier of a 3-tier (or n-tier) application, but the application has to be completed by hand coding, or only offer support for entering the result of the modeling process and do not support the modeling process itself.

6 REFLECTION AND CONCLUSION

We presented a new area for applied research: fully automated generation of Information Systems applications.

Information Systems Development can greatly profit from established research results of the Functional Programming Languages field. Up until now, data modeling - the heart of ISD - is carried out with implementation oriented languages with a (very) limited expressive power and without proper abstraction mechanisms.

The family of Fact Oriented Modeling methods is a promising starting point for integrating FPL research results into data modeling methods and tools, because the strong type systems of FPL's naturally fit into their basic data structuring approach. On basis of this integration of types into the method, the full accomplishments of FPLs, for example generic programming, could eventually become available.

Interested researchers should be aware of the peculiarities of the ISD field.

The intended target group (IS developers and stakeholders of IS applications) is in general not educated in Computer Science, has little or no experience with formal methods, and does barely recognize the benefits of an engineering approach. A classical academic research approach will result in methods and tools hardly useable by this group - more than in many other fields. Doing successful research in the ISD field is therefore not only a academic challenge but also a social one.

REFERENCES

[1] Jennifer Stapleton and Peter Constable. *DSDM : A framework for business centered development*. Addison-Wesley, 1997.

[2] Edward Yourdon. *Modern Structured Analysis*. Prentice Hall, 1988.

[3] Jan L. Harrington. *Relational Database Design Clearly Explained, Second Edition*. Morgan Kaufmann, 2002.

[4] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Addison-Wesley, 1991.

[5] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*. Addison-Wesley, 2003.

[6] Edward Yourdon. *Death March, Second Edition*. Prentice Hall, 2003.

[7] http://www.wikipedia.org/. Article ”Object-Relational impedance mismatch”. Using the search *Object-Relational impedance mismatch*.

[8] G. M. Nijssen and Terry Halpin. *Conceptual Schema and Relational Database Design*. Prentice Hall, 1989.

[9] T. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2001.

[10] Guido Bakema, Jan Pieter Zwart, and Harm van der Lek. *Volledig Communicatiegeorinteerde Informatiemodellering FCO-IM*. Academic Service, The Netherlands, 2005. Textbook in Dutch. The English version can be downloaded from http://www.casetalk.com/php/index.php?FCO-IM%20English%20Book.

[11] Arthur H. M. ter Hofstede, Henderik Alex Proper, and Theo P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, 1993.

[12] Terry Halpin. ORM 2. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, volume 3762 of *LNCS*, pages 676–687, 2005.

[13] T. Halpin, K. Evans, P. Hallock, and B. Maclean. *Database Modeling with Microsoft Visio for Enterprise Architects*. Morgan Kaufmann, 2003.

[14] http://www.casetalk.com. CaseTalk website.

[15] Eric John Pierson and Necito dela Cruz. Using Object Role Modeling for Effective In-House Decision Support Systems. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, volume 3762 of *LNCS*, pages 636–645, 2005.

[16] Arthur H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, The Netherlands, 1993.

[17] Marlon Dumas, Lachlan Aldred, and Arthus ter Hofstede. From Conceptual Models to Constrained Web Forms. In V. Kashyap and L. Shklar, editors, *Real-World Semantic Web Applications*. IOS Press, 2002.

[18] Terry Halpin. Modeling Collections in UML and ORM. In K. Siau, editor, *Proc. EMMSAD00: 5th IFIP WG8.1 Int. Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, 2000.

[19] T.A. Halpin and H.A. (Erik) Proper. Subtyping and Polymorphism in Object–Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[20] G.P. Bakema, J.P.C. Zwart, and H. van der Lek. Fully communication oriented NIAM. In *NIAM-ISDM 1994 Conference, Working Papers*, pages L1–L35, 1994.

[21] Rinus Plasmeijer and Marko van Eekelen. *Concurrent CLEAN Language Report (version 2.0)*, December 2001. http://www.cs.ru.nl/∼clean/.

[22] Betsy Pepels and Rinus Plasmeijer. Generating Applications from Object Role Models. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, volume 3762 of *LNCS*, pages 656–665, 2005.

[23] Johan Jeuring and Rinus Plasmeijer. Generic Progamming for Software Evolution. In *http://w3.umh.ac.be/evol*, 2006.

[24] M.R.C. Pil. Dynamic types and type dependent functions. In Kevin Hammond, Tony Davie, and Chris Clack, editors, *Implementation of Functional Languages (IFL '98)*, volume 1595 of *LNCS*, pages 169–185. Springer Verlag, 1999.

[25] Peter Achten, Artem Alimarine, and Rinus Plasmeijer. When generic functions use dynamic values. In R. Peña, editor, *The 14th International workshop on the Implementation of Functional Languages, IFL'02, Selected Papers*, volume 2670 of *LNCS*, pages 17–33. Madrid, Spain, Springer, September 2002.

[26] Rinus Plasmeijer and Peter Achten. iData For The World Wide Web - Programming Interconnected Web Forms. In *Proceedings Eighth International Symposium on Functional and Logic Programming (FLOPS 2006)*, volume 3945 of *LNCS*, Fuji Susono, Japan, Apr 24-26 2006. Springer Verlag.

[27] Rinus Plasmeijer and Peter Achten. The Implementation of iData - A Case Study in Generic Programming. In A. Butterfield, editor, *Proceedings Implementation and Application of Functional Languages, 17th International Workshop, IFL05*, Dublin, Ireland, September 19-21 2005. Technical Report No: TCD-CS-2005-60.

[28] Daan Leijen and Erik Meijer. Domain specific embedded compilers. In *2nd USENIX Conference on Domain Specific Languages (DSL'99)*, pages 109–122, Austin, Texas, October 1999. Also appeared in ACM SIGPLAN Notices 35, 1, (Jan. 2000).

[29] Jérôme Siméon and Philip Wadler. The Essence of XML. In *POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–13, New York, NY, USA, 2003. ACM Press.

[30] http://www.omg.org/mda/. MDA website.

[31] Mark G. J. van den Brand, J. Heering, P. Klint, and P. A. Olivier. Compiling language definitions: the ASF+SDF compiler. *ACM Trans. Program. Lang. Syst.*, 24(4):334–368, 2002.

[32] Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Transactions on Software Engineering Methodology*, 14(3):331–380, 2005.