Chapter 4
Towards P Systems Based Approach for Evolutionary Enterprise Application

Gunnar Piho, Jaak Tepandi, and Viljam Puusep

Abstract Development of enterprise applications is expensive, takes time and requires knowledge, tools and techniques. Contemporary enterprise applications must be dependable as well as customizable in the evolutionary way according to changes in the enterprise business processes. The wider goal of our research is to develop techniques for development of enterprise applications that software end users, in collaboration with software developers, are able to change safely and easily according to changing requirements. In accordance to the software engineering triptych: to write software, the requirements must be prescribed; to prescribe the requirements, the domain must be understood; to understand the domain, we must study one. We present and exemplify P systems based enterprise domain model. We treat an enterprise as a membrane-computing structure and utilize P system notions, notations and formalisms in modelling of enterprises and enterprise business processes. In our understanding this P systems based enterprise model can provide a practically usable framework for development of evolutionary enterprise applications.

Keywords Evolutionary Enterprise Applications • P Systems • Model Driven Engineering • Laboratory Information Management Systems

G. Piho (✉)
Clinical and Biomedical Proteomics Group, University of Leeds,
Beckett St, Leeds LS9 7TF, UK

Department of Informatics, Tallinn University of Technology,
Akadeemia St. 15A, Tallinn 12618, Estonia
e-mail: g.i.piho@leeds.ac.uk; gunnar.piho@ttu.ee

J. Tepandi • V. Puusep
Department of Informatics, Tallinn University of Technology,
Akadeemia St. 15A, Tallinn 12618, Estonia
e-mail: jaak.tepandi@ttu.ee; viljam.puusep@ttu.ee

M. José Escalona et al. (eds.), *Information System Development: Improving Enterprise Communication*, DOI 10.1007/978-3-319-07215-9_4,

4.1 Introduction

The goal of our research is to develop practically usable techniques for development of enterprise applications that software end users, in collaboration with software developers, are able to change safely and easily according to changing requirements. We use a case-study-based research methodology. The case is Laboratory Information Management System (LIMS) software development in Clinical and Biomedical Proteomics Group (Cancer Research UK Clinical Centre, Leeds Institute of Molecular Medicine, St. James University Hospital at University of Leeds). LIMS represents a class of computer systems designed to manage laboratory information [1].

According to *software engineering triptych*, in order to develop software we have to: (a) informally and/or formally describe a domain (D); (b) derive requirements (R) from these domain descriptions; and (c) finally from these requirements we have to determine software design specifications and implement the software (S), so that $D, S \vDash \mathscr{R}$ (meaning the software is correct) holds [2]. The term *domain* or *application domain* can be anything to which computing can be applied [3]. In our studies the application domain is business domain in general (producing, buying and selling either products or services) and clinical research laboratory domain in particular.

In research laboratories, like CBPG, business processes are changing constantly and different research groups within the same research laboratory, sometimes even different investigators in one and the same research group, require different business processes and different or differently organized data. While standardized in some ways, such system for scientists has to be flexible and adaptable so, that there are customizable possibilities to describe data, knowledge and also research methods.

In addition to the three principal layers (presentation, domain and data source), in contemporary distributed enterprise application architecture there is also a communication (containing and connecting logic) layer [4, 5], illustrated on Fig. 4.1.

In our understanding the communication layer and the presentation layer are similar in their nature. The presentation layer gives humans an interface (forms, documents, etc.) to the defined logic (domain model). Similarly, the communication layer gives artificial agents (services, software systems, etc.) an interface (communication protocols, etc.) to the defined logic.

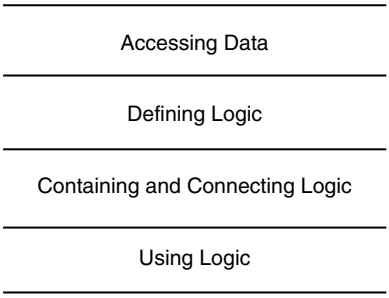

Fig. 4.1 4-Tier architecture of distributed enterprise application architecture

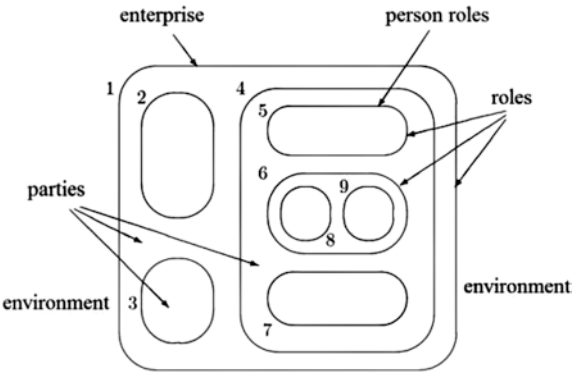

Fig. 4.2 A cell-like enterprise structure

We are looking for ways to minimize (better to completely avoid) changes in the domain logic and in the data source (and therefore also in data access) layers as these changes are risky and time consuming. We are trying to find possibilities to fulfil user requirements only by making changes in the presentation or in the communication layers. It would be nice if these changes can be made by end users even at run-time.

P system (membrane computing) [6] is the model for the distributed computing where multisets of symbol objects encapsulated into membrane structure are processed in the maximum parallel manner. P system aims to mimic the structure and the functioning of living cells. In [7] we presented and exemplified P systems based enterprise domain model. We treated an enterprise as a membrane-computing structure (Fig. 4.2) and utilized P system notions, notations and formalisms in modelling of enterprises and enterprise business processes.

In the current paper we describe, how in our understanding this P systems based enterprise domain model can provide a practically usable framework for development of evolutionary enterprise applications.

In Sect. 4.2 we describe the related works. In Sect. 4.3 we shortly describe the P systems based enterprise domain model we presented in [7]. We propose and illustrate the evolutionary criteria and explain the possible UI based evolution of the enterprise applications in Sect. 4.4. These criteria are derived from the P systems based enterprise domain model. We conclude in Sect. 4.5.

4.2 Related Works Towards Evolutionary Information Systems

Searching in the document titles of the IEEE Xplore digital library the words “evolution”, “information” and “system”, we found 30 papers. Eight of them we found were related to the evolutionary information systems research topic.

Layzell and Luocopoulus [8], in 1988, described the collaborative European project RUBRIC, whose aim was to improve the practice of constructing large, sophisticated information systems by avoiding the practice of embedding business policy within program logic. The target was to separate not only business data, but also business policy from the operational procedure. They proposed a rule-based approach to the construction and evolution of business information systems. The structural components were based on entity-relationship model consisting of entities, which are any concrete or abstract things from the universe of discourse.

Clark, Lobsitz and Shields [9], in 1989, described the documenting process of the TASC-EDGETM (The Analytic Science Corporation—Effective Development through Growth and Evolution) methodology for the evolutionary software development. Based on the classical waterfall model and on the manual change of the code, the main models (process model, information model, and development model) of the TAS C-EDGE methodology are eligible also today. By combining development, operation (includes user feedback) and requirements analysis this TASC-EDGE evolutionary software development was based on the principle of listening to the system users and to responding to their needs.

Oei, Proper and Falkenberg [10], in 1992 *discussed the need for information systems capable of evolving to the same extent as organization systems do* and proposed a formal model for the evolution of information systems. Informally this model transformed the requests from users of the information system into update and retrieval actions of the application.

In 1992, Lui and Chang [11] were working for evolutionary information systems. They proposed a Visual Specification Model for specification design, change and redesign. In this method they focussed on the database schema changes in a single relation and on their effects to other components in the specification. The wider goal of their research was to develop a unified methodology to construct conversion functions, to maintain change history, and to detect inconsistency when multiple relations involved in a change.

Shifrin, Kalinina and Kalinin [12], in 2002, described the MEDSET technology for development, deployment and support of information systems in clinical medicine and other poorly formalized subject domains. The basic statements they pointed out were: (a) realization data structures in compliance with the structure of business process; (b) stability of the database model; and (c) user interface consistency to support some definite part of one business process.

Wang, Liu and Ye [13], in 2008, pointed out the potential inconsistencies among the ontology and the dependant applications in case of the ontology evolution and analysed the approaches of maintaining the consistency and keeping the continuousness of the dependant applications during the evolution. They proposed two scenarios for maintaining the consistency: property split and property range changes. By splitting they mean the situation, when for instance the text property of employee's home address is split into more specific properties of city, district, street, etc. In splitting the business process itself does not change: only the data used in this business process becomes more accurate. By range changes they mean the situation when the business process itself will change. For example the funding: what was only for full time students is now available also for part time students.

Aboulsamh and Davies [14], in 2010, proposed a metamodel-based approach to information systems evolution and data migration. They expand model-driven engineering to facilitate the evolution of information systems. They claim that in the domain of information systems, after the initial development and delivery, the automatic change of the systems is useful only if the data held in the system can be migrated easily to the next version and therefore their proposed method is focussed on migration of the information held in the system data.

Ralyté, Arni-Bloch and Léonard [15], in 2010, proposed a process model for integrating new services with the information system. They claim, that *before publishing a service to be reused in some composition, the validation of the data consistency, rules soundness, process compatibility and organizational roles compliance have to be guaranteed.*

4.3 P Systems Based Enterprise Domain Model

Cells and enterprises have both very intricate structure and functioning. Similarities can be seen in their internal activities as well as in their interactions with the neighbouring cells/enterprises, and with the environment in general.

Both cells and enterprises have a way to organize and control their processes. Cells have biological processes, enterprises have business processes and both have informational processes. Any cell means membranes. Any enterprise means enterprise structure. Both cells and enterprises are clearly separated from their environment and both have compartments (membranes/enterprise structure units) where specific processes take place.

Similarities can be seen also in sending messages from compartment to compartment. Cells can transport packages of molecules from one part to other part through vesicles enclosed by membranes in a way that transported molecules are not "aggressed" during their journey by neighbouring chemicals. Enterprises have internal or use external post and transportation services to transport messages, documents and products from location to location in a secured way.

Similarities can be seen also in creating and destroying compartments. Cell is able to create and destroy membranes in order to make biochemical processes more efficient. Enterprise is able to create and liquidate their structural units in order to make business processes more efficient.

In summary, cells and enterprises contain many fascinating similarities at various levels starting with similarities in structure, in communication/cooperation among neighbouring cells/enterprises, and ending with the interaction with the environment.

In Fig. 4.2 the membrane like enterprise structure is illustrated. Membranes are the roles [16] the parties (persons or enterprise units) can play in order to fulfil some enterprise related tasks. The external membrane (skin) we call enterprise. The elementary membrane is the role only persons can play. Patients, managers, doctors, students, etc. are the examples of such person only roles. The regions in the

context of enterprise are parties (persons and/or organizations—groups of persons) playing the particular role. In P systems based enterprise model (differently from the P systems) the distinction must be made between the notion role (membrane) and the notion party (region), because for instance, the manager's role and Mr John Smith being the manager from the date to the date do not have the same meaning. However, sometimes we can use these notions also interchangeably because of one-to-one correspondence; especially in case of roles only the organizations (e.g. hospital, laboratory, etc.) can play.

Every structural unit in an enterprise receives documents (d) (information received in speech can also be modelled as a document). In every structural unit there are also descriptions of these documents i.e. formats (*f*) of these documents. For instance, a human resources (HR) department knows exactly what should be the format (f_i) of a document (d_i) to compose an employment contract document (d_j) according to the contract document format (f_j). HR department also knows, what kind of information (a_i, e.g. name, date of birth, address, etc.) should be recorded into the company records (d_k). Therefore the business rules for the HR department can be notated as follows.

$$d_i f_i f_j \rightarrow d_j f_i f_j a_i$$
$$a_i f_k \rightarrow d_k f_k$$

This means that according to a document d_i (e.g. order), the recruitment of an employee is started. As a result the employment contract (d_j) is concluded and some employee data (a_i) are created. The employee data is then recorder into a specific document d_k (e.g. record in a list of workers) of the company. Both of illustrated rules are catalytic rules where the document formats f_i, f_j and f_k are catalysts describing precisely the documents (d_i, d_j, d_k) the company uses. This means, that documents and therefore document formats are company specific. The employee data *ai* (e.g. archetypal domain model describing persons) we consider to be written in universally understandable language of business archetypes and archetype patterns. We describe them in Sect. 4.4.1.

We can also use indicators h*ere*, i*n* and *out*. For instance the same rules with these indicators can be as $d_i f_i f_j \rightarrow (d_j, out) f_i f_j$ (a_i, *here*) and $a_i f_k \rightarrow (d_k, out) f_k$. This means for example that documents d_j and d_k created by HR department are going to the "surrounding" enterprise unit.

Membrane dissolving (δ) in P system based enterprise model means either concentrating of the post or eliminating of the structural unit of organization. Differently from the basic P system, in the P system based enterprise domain model, in case of dissolving, not only the objects and child members, but also the rules are left free in the surrounding enterprise unit. We also expect that in P system based enterprise domain model the enterprise itself (skin membrane) can be dissolved.

Similarly to base P system, also in the P system based enterprise model the rules are processed non-deterministically in the maximally parallel manner. Non-determinism can be seen for instance as HR department's possibility to choose the order of rules (creating employment contract or recording employee data) and objects (whatever document first) deliberately.

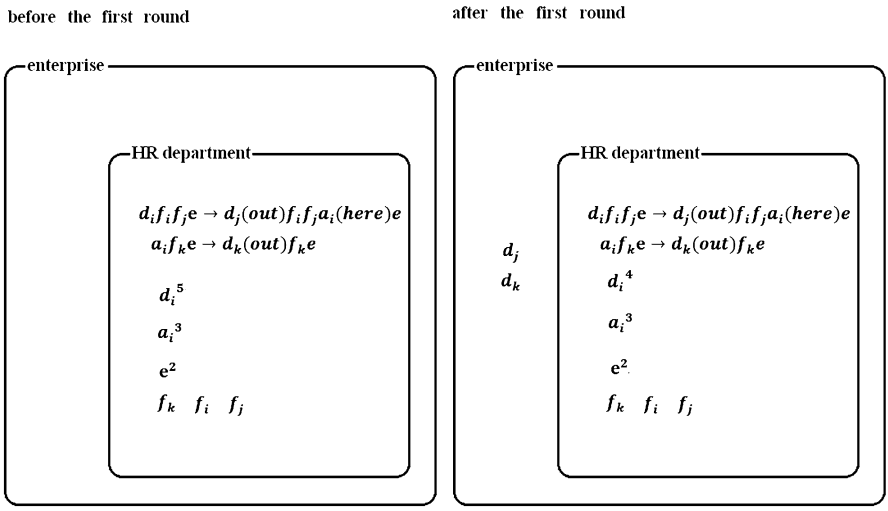

Fig. 4.3 Processing of rules in P systems based enterprise domain model

For the maximum parallelism we probably have to determine the employees (catalyst e) in the HR department rules, e.g. $d_i f_i f_j e \rightarrow d_j(out) f_i f_j a_i$ $(here) e$ and $a_i f_k e \rightarrow d_k(out) f_k e$.

For instance (illustrated in Fig. 4.3), in case there are five orders (d_i^5), three units of employee data (a_i^3) and two HR department employees (e^2) in HR department, then after the first round of processing with the maximum parallel manner the HR department objects can be $d_i^3 a_i^5 e^2$ (both employees were choosing the first rule), $d_i^5 a_i^1 e^2$ (both were choosing the second rule) or $di^4 ai^3 e^2$ (shown in picture, one employee choose the first rule and other the second rule).

Concluding the paper [7], in our understanding, we can formally define the enterprise using the formal definition ($\Pi = (O, C, \mu, \omega_1, \omega_2, \ldots, \omega_m, R_1, R_2, \ldots, R_m, i_0)$ of P systems of degree m. Looking for the Fig. 4.1, the enterprise application architecture (also some model of the enterprise) we have *data* ($\omega_1, \omega_2, \ldots, \omega_m$ in formal definition where ω_i means particular set of data in particular compartment, e.g. in HR department, Fig. 4.3); finite set of *classes* (O in formal definition) describing business domain logic objects (describing any data) and workflows (using the data); different business domain *rules* ($R_1, R_2, \ldots, R_m$ in formal definition where *Ri* means particular set of rules in particular compartment, Fig. 4.3); different *document formats* (catalysts, C) for converting the data defined by business domain logic to and from man or machine readable documents like printable reports, web pages, standalone client forms, communication protocols for remote logic (another computer or software) etc. We also have the enterprise structure (μ) and the compartment (i_0) which contains (can contain) the result of the calculation. Informally it means that any enterprise is a "computer" calculating its budget. Result of this "calculation" is kept in the accounting department (compartment, i_0) data.

4.4 Towards Evolutionary Enterprise Applications

We see the concept of P-systems $\Pi = (O, C, \mu, \omega_1, \omega_2, \ldots, \omega_m, R_1, R_2, \ldots, R_m, i_o)$, described by classes ($O$), catalysts ($C$), structure ($\mu$), data ($\omega_1, \omega_2, \ldots, \omega_m$) and rules ($R_1, R_2, \ldots, R_m$), as a roadmap towards evolutionary information systems.

In our understanding, the enterprise application is evolutionary, if software end users, in collaboration with software developers, are able to change safely and easily (according to changing requirements) the following: (1) classes (O) describing domain concepts; (2) document formats (catalysts, C) that the system uses; (3) the structure (μ) of the system: (4) data ($\omega_1, \omega_2, \ldots, \omega_m$) can be changed anyway (we can add as many lines as needed to documents or send as many documents as needed); (5) calculation rules ($R_1, R_2, \ldots, R_m$); and (6) the location (i_o) where the calculation results are held.

We describe here (because of limited number of pages) only how we change classes and document formats in current LIMS for CBPG project and let the other two for our later papers.

4.4.1 Classes Are Changed Using Archetypes and Archetype Patterns

O is the finite and non-empty set of all possible *classes* (names of concepts and groups of concepts as well as logical and computational models of these concepts and groups of concepts) used in modelling of enterprises (then names and logical models of concepts) and in enterprise application implementations (then computational models of these concepts). Let as suppose, that we have at least two subsets of classes in O: (a) classes describing archetypes and archetype patterns ($O_A \subset O$), (b) classes describing business domain logic ($O_B \subset O$; $O_A \subset O_B$).

Business archetypes and archetype patterns are originally designed and introduced by Jim Arlow and Ila Neustadt [16]. Business archetype patterns (product, party, order, inventory, quantity and rule), composed by business archetypes (person's name, address, phone number, etc.) are information models and describe the universe of discourse of businesses as it is, neither referring to the software requirements nor to the software design. For instance, in Fig. 4.4 the *party archetype pattern* is illustrated.

The party archetype pattern represents a (identifiable, addressable) unit that may have a legal status and has some autonomous control over its actions. Persons and organizations are types of parties. Party has different properties like *party contacts* (phone number, e-mail, web address, and postal address), *registered identifiers* (passport, VAT number, domain name, stock exchange symbol, etc.) etc. Each party can play different *roles* (patient, clinician, employee, customer, etc.) and can be in different *relationships* (e.g. being married, having a child, studying in school). Both roles and relationships are time limited (property *valid* in role and

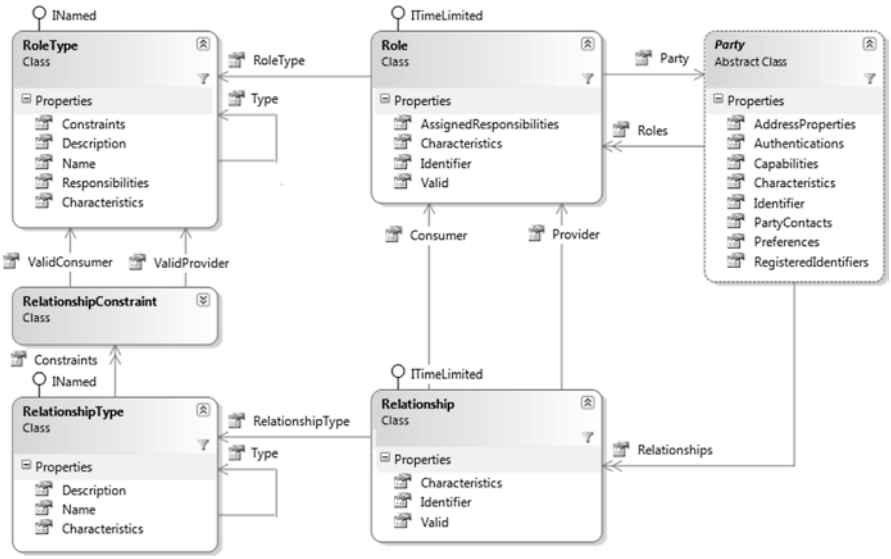

Fig. 4.4 Party archetype pattern

relationship archetypes). *Role type* is used to store common information (e.g. constraints describing parties who can play the role) for a set of similar role instances and *relationship type* is used to store common information (e.g. constraints describing which roles can form the relationship) for a set of a similar relationship instances. In the current model only binary (more flexible and cleaner than n-ary) relationships are used, which means that one *relationship* binds exactly two related roles conditionally called as "provider" and "consumer".

Let us suppose, that classes illustrated in Fig. 4.4 are realized in code, and that we also have a database schema support for such classes (this is what we have in CBPG LIMS project). This means, that we have database tables for recording *party* instances (persons and organizations), *role* instances, *relationship* instances, *relationship type* instances and *role type* instances.

In a clinical laboratory, for instance, the common roles for persons are patient (whose blood will be tested), clinician (who ordered the blood testing), and medical technical assistant (MTA, who performed the blood testing). Normally in the domain logic layer of enterprise applications in such case we have to realize *Patient*, *Clinician* and MTA classes in code. In archetypes and archetype patterns based approach, we use, we do not. We have only one *RoleType* class for all classes and one *Role* class for all possible class instances (e.g. objects) of all possible classes. Such an approach gives for us the possibility to add new "classes" even at runtime. For instance, to add a new "class" named *laboratory manager*, the only thing to do is to add a new record to the *RoleType* database table.

To "add new properties" in such an archetypes and archetype patterns based approach, we have the property *Characteristics* (Fig. 4.4) in all our programmed

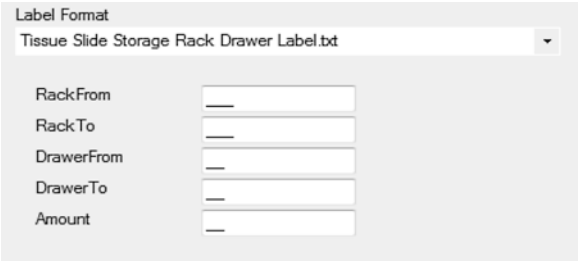

Fig. 4.5 Fragment of generated barcode printing dialog

Fig. 4.6 Example of generated barcode

classes. These are constructs similar to the *RDF* (Resource Description Framework) *triplets*. This means, that the *Characteristics* property holds a collection of {*category*, *name*, *value*, *authorized by*, *valid from*, *valid to*} records. For example the notation "person is 176 cm tall, measured by Dr Smith at 3rd of May 2000" is a characteristic with "body metrics" denoting *category*, "is tall" denoting *name*, "176 cm" denoting *value*, "Dr Smith" denoting *authorized by*, and "3rd of May 2000" denoting *valid from*.

4.4.2 Document Formats Are User Editable

We use document formats (e.g. end user editable files, DB records or values of properties) in number of places in current LIMS for CBPG project. For example, the following end user editable script (content of a file)

$$
\begin{aligned}
&;\{a\}\ SLR\\
&;\{b\}\ I\ RackFrom = 990\\
&;\{c\}\ I\ RackTo = 990\\
&;\{d\}\ C\ Rack = Foreach(b, c)\\
&;\{e\}\ I\ DrawerFrom = 90\\
&;\{f\}\ I\ DrawerTo = 90\\
&;\{g\}\ C\ Drawer = Foreach(e, f)\\
&;\{0\}\ C\ Barcode = a{:}3|d{:}3|g{:}2\\
&;\{1\}\ I\ Amount = 90
\end{aligned}
$$

first describes the automatically generated dialog, shown in Fig. 4.5, and then prints the barcode (Fig. 4.6) according to entered, using this dialog, values.

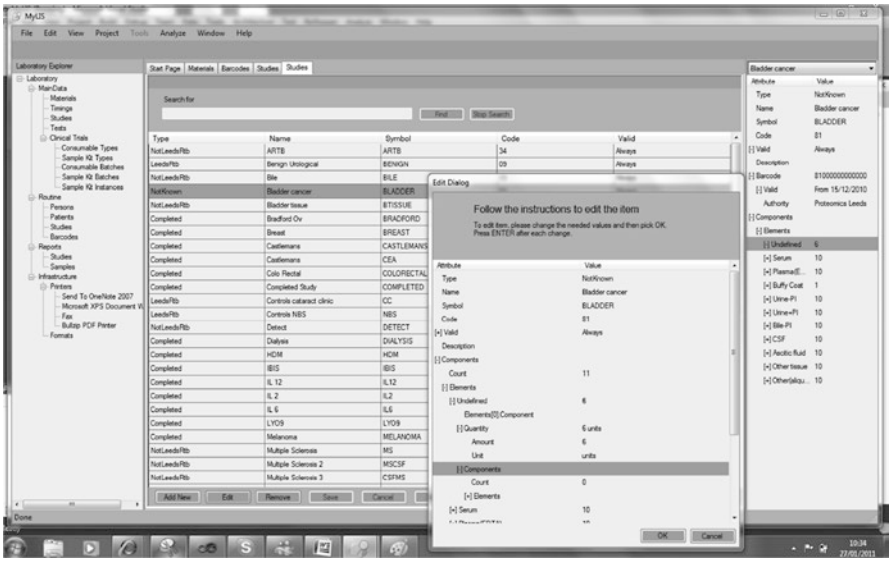

Fig. 4.7 Screenshot of MyLIS (LIMS for CBPG) user interface

The other example is how we generate UI at runtime. For example, the UI, illustrated in Fig. 4.7, is generated according to the following scripts.

```
static string[] GridRange = new[] { "Type","Name","Symbol","Code","Valid" };
static string[ ]PropertyRange = new[] {"Type","Name","Symbol","Code",
    "Valid","Description","Barcode","Components" };
static string[] EditRange = new[] {"Type","Name","Symbol","Code","Valid",
    "Description","Components" };
```

First, the *GridRange* lists the properties, by their names, the master grid shows. Next, the *PropertyRange* lists the properties the detail panel (left side panel of main form) shows. Finally, the *EditRange* lists the properties the edit dialog shows.

This means that we have document formats (GridRange, PropertyRange, EditRange) which describe documents (user interfaces). When we change document formats, the user interfaces, and therefore the information system, will change. As document formats are properties, it is possible to change the values of these properties at runtime using, for example, reflection technology. Generally speaking we use the document formats as catalysts (C) for mapping data described by business domain logic classes to user or machine readable documents. Different mappings require varying levels of knowledge and authority from the end users. Therefore an organisation deploying such a system should determine the ability of users to perform a specific mapping, for example by specifying their roles and responsibilities with the aid of a RACI (Responsible, Accountable, Consulted, Informed) matrix.

4.5 Conclusion

We see the concept of P-systems ($\Pi = (O, C, \mu, \omega_1, \omega_2, \ldots, \omega_m, R_1, R_2, \ldots, R_m, i_o)$), described by objects (O), catalysts (C), structure (μ), data ($\omega_1, \omega_2, \ldots, \omega_m$) and rules ($R_1, R_2, \ldots, R_m$), as a roadmap towards evolutionary information systems. We follow the software engineering triptych: to write software, the requirements must be prescribed; to prescribe the requirements, the domain must be understood; to understand the domain, we must study one.

We presented and exemplified the P systems based enterprise domain model. We treated an enterprise as a membrane-computing structure and utilized P system notions, notations and formalisms in modelling of enterprises and enterprise business processes.

The wider goal of our research is to develop techniques for development of enterprise applications that software end users, in collaboration with software developers, are able to change safely and easily according to changing requirements.

In our understanding this P systems based enterprise domain model can lead us towards evolutionary enterprise applications. In our understanding, the enterprise application is evolutionary, if software end users, in collaboration with software developers, are able to change safely and easily (according to changing requirements) the following:

1. Classes (O) describing domain concepts. (For example being able to define a new domain concept *patient*, using some archetypal language (Sect. 4.4.1), so that *patient* is a *role* that only *party* who is a *person* (Fig. 4.4) can play).
2. Document formats (catalysts, C) that the system uses.
3. The structure (μ) of a system.
4. Data ($\omega_1, \omega_2, \ldots, \omega_m$) can be changed anyway. We can add as many lines as needed to documents or send as many documents as needed.
5. Calculation rules ($R_1, R_2, \ldots, R_m$).
6. Location (i_o) where calculation results are held.

Acknowledgments This work is supported by Estonian Ministry of Education and research (SF0140013s10); by Tallinn University of Technology (Estonia); by University of Leeds (United Kingdom); by Cancer Research UK.

References

1. ASTM (2006) E1578-06 Standard guide for laboratory information management systems (LIMS). ASTM International. http://www.astm.org/Standards/E1578.htm. Accessed 4 June 2014
2. Bjørner D (2007) Domain theory: practice and theories (a discussion of possible research topics). Macau SAR, China. http://www.imm.dtu.dk/~dibj/ictac-paper.pdf. Accessed 4 June 2014
3. Bjørner D (2006) Software engineering. In: Abstraction and modelling. Texts in theoretical computer science. The EATCS series, vol 1. Springer, Heidelberg
4. Fowler M (2003) Patterns of enterprise application architecture. Addison-Wesley, Boston, MA

5. Chappell D (2006) Comparing .NET and Java: the view from 2006. Microsoft TechEd Developers, Barcelona
6. Paun G (2004) Introduction to membrane computing. http://psystems.disco.unimib.it/download/MembIntro2004.pdf. Accessed 30 Aug 2011
7. Piho G, Tepandi J, Puusep V (2013) P systems based enterprise domain model. Research Report. Tallinn University of Technology
8. Layzell P, Loucopoulos P (1988) A rule-based approach to the construction and evolution of business information systems. In: Software maintenance
9. Clark P, Lobsitz R, Shields J (1989) Documenting the evolution of an information system. In: Aerospace and Electronics Conference, NAECON 1989. Proceedings of the IEEE 1989 National
10. Oei J, Proper H, Falkenberg E (1992) Modelling the evolution of information systems. Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands
11. Liu C, Chang S (1992) A visual specification model for evolutionary information systems. In: Computing and information, 1992. Proceedings. ICCI '92
12. Shifrin M, Kalinina E, Kalinin E (2002) MEDSET – an integrated technology for modelling, engineering, deployment, support and evolution of information systems. In: Computer-based medical systems 2002 (CBMS 2002)
13. Wang Y, Liu X, Ye R (2008) Ontology evolution issues in adaptable information management systems. In: e-Business engineering, 2008. ICEBE '08
14. Aboulsamh M, Davies J (2010) A metamodel-based approach to information systems evolution and data migration. In: Software engineering advances (ICSEA), 2010 fifth international conference
15. Ralyté J, Arni-Bloch N, Léonard M (2010) Information systems evolution: a process model for integrating new services. In: AMCIS 2010 proceedings
16. Arlow J, Neustadt I (2003) Enterprise patterns and MDA: building better software with archetype patterns and UML. Addison-Wesley, Boston, MA