Usage Models Mapped to Programs

András J. Molnár[1,2][0000−0002−2194−0320] and
Bernhard Thalheim[1][0000−0002−7909−7786]

[1] Christian-Albrechts-University Kiel, Computer Science Institute,
D-24098 Kiel, Germany
{ajm,thalheim}@is.informatik.uni-kiel.de
[2] MTA-SZTAKI Institute for Computer Science and Control,
Hungarian Academy of Sciences, H-1111 Budapest, Hungary
modras@ilab.sztaki.hu

Abstract. Model-based programming can replace classical programming based on compilation and systematic development of models as well on explicit consideration of all model components without hiding intrinsic details and assumptions. A key element of model-based programming is the proper definition and management of model suites, by which multiple, interrelated models can be transformed from one another and their consistency is ensured after modifications. A usage model is based on the specification of user roles and types, together with an interaction space described in a form of a storyboard, showing which activities are supported, in which order, by which actors. A workflow model is an extended, well-formed declaration of how specific processes should be carried out. It can directly be translated to program code, using a proper workflow or process engine. A novel way of programming is being opened up by usage modeling, which is being investigated in this paper: given a storyboard with supported usage scenarios, it is possible to derive a workflow model from it. We present our two translation methods using a working example, identifying guidelines as requirements for model refinement and normalization, rules for model translation, and propose considerations towards improved methods and model specifications.

Keywords: Model-centered programming · Model to program · Model suite · Model transformation · Storyboard · Process model.

1 Introduction

1.1 Programming by Modeling

Programming is nowadays a socio-technical practice in most disciplines of science and engineering. Software systems are often developed by non-programmers or non-computer scientists, without background knowledge and skills, or insight into the culture of computer science, without plans for systematic development. Maintenance, extension, porting, integration, evolution, migration, and modernisation become an obstacle and are already causing problems similar to the software crisis 1.0, since such systems often have a poor structure, architecture,

documentation, with a lost insight of specific solutions. Programs of the future must be understandable by all involved parties and must support reasoning and controlled realisation and evolution at all levels of abstraction.

Our envisioned *true fifth generation programming* [14] is a new programming paradigm where models are essentially programs of next generation and models are translated to code in various third or fourth generation languages. Programming is done by model development, relying on the compilation of these models into the most appropriate environment.

Application engineers and scientists are going to develop and use models instead of old-style programming, supported by templates from their application area. They can thus concentrate on how to find a correct solution to their problems, managing the complexity of software intensive systems. The process will be supported by model-backed reasoning techniques, as developers will appreciate and properly evaluate the model suite at the desired level of abstraction.

1.2 Usage Models and Workflow Models

In our study we are considering the case of web information system development.

A *usage model* of a web-is consists of specification of user roles and types, their associated goals and tasks, and an interaction space. The latter can be expressed as a graph, called a *storyboard*, describing what activities are supported and in which possible order, by which actors [9]. Supported interaction playouts can be formulated as *scenarios* (exact graph paths), *story algebra expressions* (path scemata), or more generally, subgraphs of the storyboard, including actor-specific views. The usage model is developed by a global-as-design approach.

A *workflow model* is an extended, well-formed declaration of how specific processes should be carried out, in a notation that is readily understandable by all stakeholders, including business analysts, technical developers and people who manage and monitor those processes [7]. A de facto standard is *BPMN* [7], but it is possible to use another workflow description language. The workflow model can directly be translated to software process components, using a proper workflow or process engine (e.g. [1]). This opens up a novel way of programming by usage modeling, via intermediate translation to a workflow model.

1.3 Related Work

Our current contribution can be related – amongst others – to the following previous works. Notions of *models* are discussed in [12]. [11] introduces *model suites* consisting of multiple, explicitly associated models, where the association uses maintenance modes, similar to integrity support in databases [16]. The *models as programs – true fifth generation programming* agenda is proposed in [14]. For data structuring, translation of entity-relationship models to relational database schemata is well-known [3, 13]. We are proposing a similar approach for the dynamics of functionality, motivated by compilers [8]: phases of preprocessing, parsing and syntax checking is followed by semantic analysis resulting an intermediate structure, and finally a possible optimization phase of the resulting,

translated model. [10] discusses *proceses-driven applications* and *model-driven execution* in terms of BPMN [7] diagrams. [2] elaborates a generative approach to the functionality of interactive information systems. [15] introduces dynamically combinable *mini-stories* to handle workflow cases with large flexibility. Although these latter works consider steps and ideas we can apply here, our currently addressed problem of usage model translation to workflow model is not explicitly discussed in any of the publications known to us.

1.4 Goal and Outline of the Paper

Our general vision is to generate running program code based on a usage model specification. We investigate on a particular sub-case in this paper: given a usage model as a storyboard with supported scenarios [9], is there a formalizable method to derive a workflow model in BPMN [7] from it.

We present our proposed path and general framework for modeling as next generation programming in Section 2, based on [14]. Section 3 introduces our target case of workflow model elicitation from a usage model, illustrated by a working example, with general guidelines for model refinement and enhancement, rules and two different methods for translation. We conclude and close with future issues in Section 4.

2 Modeling and Programming

Models are universal instruments for communication and other human activities. Ideas and thought chunks can be presented to those who share a similar culture and understanding without the pressure to be scientifically grounded. A model is an adequate (i.e. analogous, focused, purposeful) and dependable (i.e. justified, sufficient in quality) instrument that represents origins and performs functions in some deployment scenario [12]. As an instrument, the model has its own background (i.e. grounding, basis) and should be well-formed. Models are more abstract than programs, but can be as precise and appropriate as programs. They support understanding, construction of system components, communication, reflection, analysis, quality management, exploration, explanation, etc. Models can be translated to programs to a certain extent, therefore, models can be used as higher-level, abstract, and effective programs. They are, however, independent of programming languages and environments. Models encapsulate, represent and formulate ideas both as of something comprehended and as a plan. Models declare what exactly to build and can be understandable by all stakeholders involved in software system development. They become general and accurate enough, and can be calibrated to the degree of precision that is necessary for high quality [14].

2.1 Model Suites

A model suite [11] consists of a coherent collection of explicitly associated models. A model in the model suite is used for different purposes such as commu-

nication, documentation, conceptualisation, construction, analysis, design, explanation, and modernisation. The model suite can be used as a program of next generation and will be mapped to programs in host languages of fourth or third generation. Models delivered include informative and representation models as well as the compilation of the model suite to programs in host languages. Consistency can be ensured similarly to relational databases [16]. Models will thus become executable while being as precise and accurate as appropriate for the given problem case, explainable and understandable to developers and users within their tasks and focus, changeable and adaptable at different layers, validatable and verifiable, and maintainable.

2.2 The Layered Model Development Network

The approach of layering has already often and successfully been used. Most program language realisations and application development methodologies follow this approach. We assume a general layered approach as the universal basis for treatment of models as programs [14], at least for programming by non-specialists. The approach of professional programmers is, however different, can also be supported in this manner as the success of programming environments has already been demonstrating, primarily due to their ease of use. Layering has also been the guiding paradigm behind the TeX and LaTeX text processing realisations [5, 6] with a general setup layer, the content layer, the adaptable device-independent layer, and the delivery layer [14].

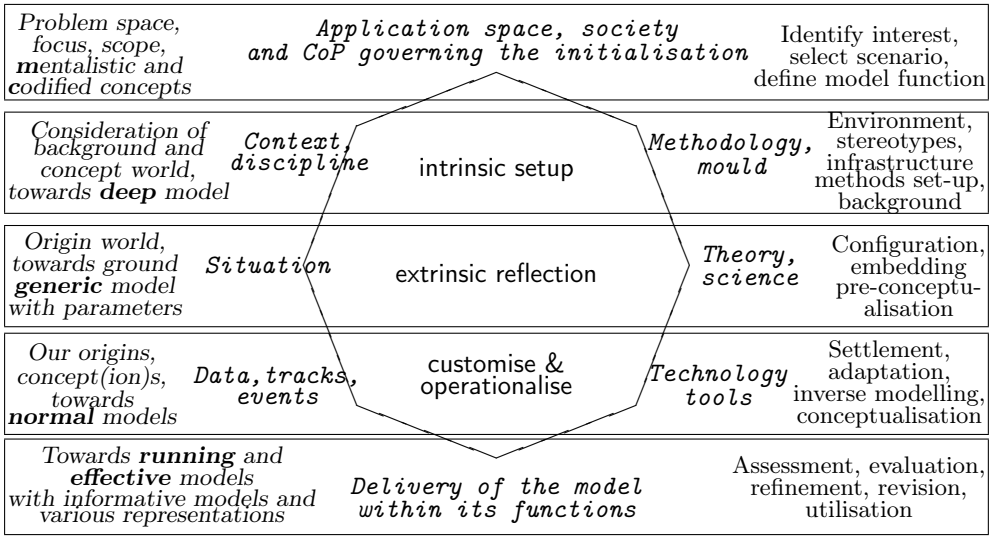

Fig. 1. The layered approach to model suite development and program generation

Model suite development and deployment will be based on separation of concern into *extrinsic* and *intrinsic* parts of models. Models typically consist on

the one side of a *normal model* that displays all obviously relevant and important aspects of a model and on the other side of a *deep model* that intrinsically reflects commonly accepted intentions, the accepted understanding, the context, the background that is commonly accepted, and restrictions for the model. The model suite will be layered into models for initialisation, for strategic setup, for tactic definition, for operational adaptation, and for model delivery (see Fig. 1).

The *initialisation layer* is given by the application and the scenarios in which models are used, by the problem characterisation, by background elements of the CoP and especially commonly accepted concepts in this community, and additionally by interest, intensions, and the value. The *enabling strategic setup layer* defines the opportunity space and especially the hidden background for the model. Its main result is the deep model that is typically assumed to be given. Normal models are not entirely developed from scratch. The *tactic definition layer* starts with some generalisation, i.e. select a ground generic model that will be customised and adapted to the normal model. The *operational customisation layer* fits, calibrates and prunes the model suite to the problem space. Finally, the *model is delivered* in various variants depending on the interest and the viewpoints of the CoP members. The complete model suite thus becomes the source for the code of the problem solution, and for the system to be built [14].

3 Elicitation of Workflow Models from Usage Models

Taking the layered approach of Fig. 1 as the general frame, we can formulate our proposed agenda in the following way: It is being declared on the initialisation layer that a website is needed for a specific application. It is analogous to selecting a *documentclass* in LaTeX, and determines the possible syntax and semantics at the underlying layers. The intrinsic, strategic setup guides the ground direction of future modeling. In our case, it will correspond to what a website means, what are the side conditions and underlying infrastructure of it and the selected application domain. It leaves open the particular modeling language or method, but gives an opportunity space and can impose requirements or proposals for the way of system development (e.g. an usage model should be developed somehow). A generic model on the tactic definition (extrinsic) layer can be, for example, a generalization of a previous storyboard development, or a configurable storyboard composed of best-practice patterns. Decision of the modeling framework or language (here, the use of storyboarding, with or without story algebra usage, in which format) must have been taken. Generic modeling must be supported by meta-models assumed to be available as (re)usable packages. Further model contents are interpreted based on the selection of these packages. The actual design is made on the operationalisation layer, forming a normal model (here: the generic storyboard is customized as needed or allowed by the generic model: missing parameters are set up, defaults can be overridden). Requirements for an acceptable normal model must have been given in the generic model or the meta-model, in order to ensure well-formedness and consistency, and to allow proper model transformations possible on the delivery layer. The normal model(s) must

be validated according to these requirements. The target, runnable model is then delivered from the normal model using a model translation, extraction or enhancement method. The target model language (here, BPMN) must be given with the selection and customization of the available translation methods. Interrelations and consistency management between the normal and the delivered model can be further declared.

3.1 An Application Case and Its Usage Model

We are assuming the development an information system for a touristic and recreational trail network, providing guidance for visitors, as well as facility management of the trails and related field assets. A map interface is being provided with planning and navigation features along the designated trails, connected to an issue tracking system for reporting and managing trail and asset defects.

The high-level usage model is given as a storyboard on Fig. 2. Abstract usage locations represented by graph nodes are called *scenes*. Users can navigate between these scenes, by performing actions associated to *directed transition links*. Some indicative *action names* are given for the reflexive links (which are in fact, denote multiple links, one by named action). The *entry point* is marked with a filled black circle. An *end point* may also be added as a double circle (by default, each scene is assumed to be a potential end point).

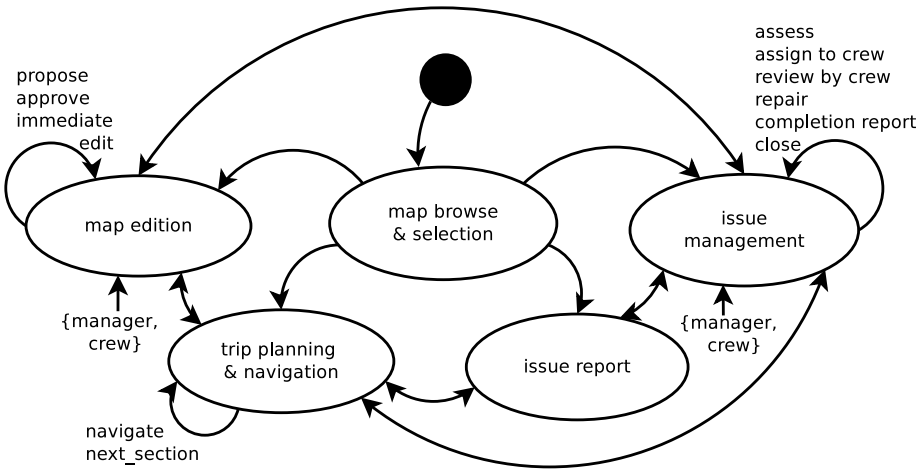

Fig. 2. High-level storyboard graph for a sample trail management system

We declare three *actor roles*: *visitor*, trail *manager* and trail *crew*. A set of authorized actors are pointed to the bottom of restricted scecnes by vertical arrows [4]. By default, all actors are allowed to enter a scene.

The storyboard is about to represent supported *normal* scenarios, as specific means the users can accomplish given tasks. Context-loosing random navigations

(e.g. back to the main page at any time) can be treated as breaking or canceling the started scenario, and starting a new scenario with a new context. These moves are not explicitly modeled so the focus can be kept on meaningful issues.

We are limiting our current discourse for one-session, one-actor scenarios.

The storyboard can be enhanced with input-output content specifications for each scene. We use the notation of [4] so that input and output content for a scene is displayed using a short horizontal arrow on the left and the right side, respectively. Input-output content is named and an output content is assumed to be delivered as an input content to the next scene along each transition link, where the content names are equal. Square brackets denote optional input or output. Content names can be prefixed by generic database operation names.

3.2 Refinement and Normalization of the Storyboard

The top-level storyboard (Fig. 2) has to be refined and enhanced, so that actual scenarios as paths in the graph will be self-descriptive and consistent, and the graph is formally sound and contains enough details for a working and meaningful translation into workflow model(s). We state the following semantical considerations and guidelines for developing the refined usage model. If all these criteria are met, and guidelines have considered, we call the storyboard *normalized*. This is only partially verifiable formally – for the items marked with (*) – and refers to a quality and stage of model development:

- Complex scenes must be decomposed into atomic sub-scenes, each having a single, well-defined action, task or activity which is fully authorized by a given set of actor roles. The interaction paths must be modeled by directed links between the sub-scenes and directly connected to outside (sub)scenes.
- Each transition link with active actor participation (action) must be replaced by a link-scene-link combination, where the action or activity is performed at the scene and the new links are only for navigation. This new scene can be handled and parametrized together with other scenes in a unified way.
- No parallel links between two scenes are allowed (*). They must either be translated using separate scenes (see above), or merged into one link, or their source or target scenes must be decomposed to separate sub-scenes.
- The routing decision (which link to follow after a scene) is assumed to be taken as part of the activity inside a scene, by default. If it is not intended, then only one outgoing link is allowed and an extra routing decision scene must be explicitly introduced after the original scene as necessary (this may be later optimized out).
- Unique names are assumed for all scenes and links (except that two or more links pointing to the same target scene can have the same name) (*).
- There must be a unique start node (entry point) with a single link to an initial scene and either a unique end node or a default rule declaring which scenes can be places for story completion (*).
- Each scene must be enhanced with a set of authorized actor roles. Without that, a default rule must be supplied. There must be no (normal) links between scenes without at least one common authorized actor role. (*)

– Input and output content is to be specified by symbolic names for each scene wherever applicable. Content names will be matched along the links (*): For each input content of a scene s there must be an output content with the same name provided by the source scene of each link directed to s. Optional content is written in square brackets.
– Input and output content names can be prefixed by database operations: SELECT is allowed for input, while INSERT, UPDATE, and DELETE are allowed for output content. Without detailed semantics of these operations, a single central application database is assumed by default.

3.3 View Generation by Actor Roles

Given a selected actor role, a specific storyboard view can be generated for it as a basis of role-specific workflow models. Unauthorized scenes for a selected role are removed with their links, resulting a cut-out of the storyboard, reachable by actors of the chosen role. An enhanced, normalized version of the visitors' storyboard view is shown on Fig. 3, with multiple sub-scenes. Links are denoted by italic numbers. An explicit end node is placed additionally, reachable from chosen scenes.

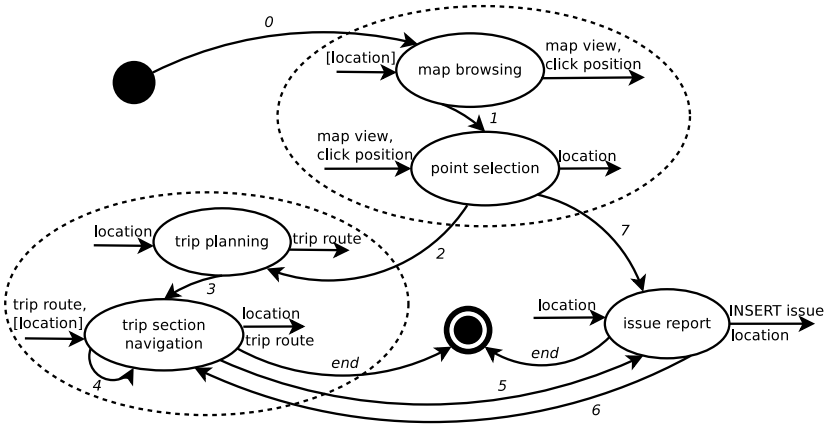

Fig. 3. Visitors' view of the refined and normalized storyboard

3.4 Graph-Based, Direct Translation Method

At this point, a default translation algorithm we have developed, can be applied to generate a BPMN process flow diagram, based on the graph connectivity of the storyboard. Details of the algorithm are omitted due to space limitations, but the result of the translation of Fig.3 is shown on Fig. 4 as a demonstrative example. The translation process can continue with enhancements indicated in Section 3.8.

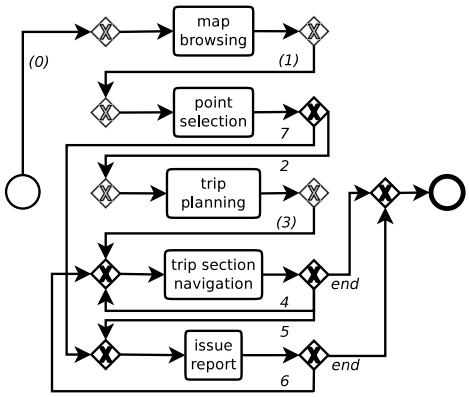

Fig. 4. Direct translation of visitor usage to BPMN, based on the storyboard graph. The full connectivity of the usage model is represented as possible process flow paths. Scenes become tasks. Numbers denote choices based on scene transition edges. Bracketed numbers are only for information, referring to original transition edges without alternatives. The grey-colored gates can be removed by merging their connections. Further refinements and optimizations are possible.

3.5 Modeling Supported Scenarios by Story Algebra Expressions

Alternatively to the previous method, a more sophisticated and targeted method is developed, if specific scenarios, which are intended to be supported by the system, are collected and expressed as patterns in a story algebra.

A particular playout of system usage becomes a path in the storyboard and is called a *scenario*. A set of possible scenarios can be modeled as using the *story algebra SiteLang* [9, p. 76], similar to regular expressions. Such a *scenario schema* can be a pattern for generating a workflow model. The original notation uses link names for description. We found that using scene names in the story algebra more naturally supports the translation to workflow models.

For example, a scenario schema of a visitor can be modeled out of the following variations: a visitor looks at the map, selects a destination point. The scenario may continue by reporting an issue for the selected point, or by planning a trip, navigating along it, and maybe at certain points, reporting an issue on-site. Each of these variants correspond to different scenarios the system should support and can be summarized as one or more scenario schemata.

Using abbreviated scene names (by first letters of words, e.g. *mb* stands for *map browse*), the above mentioned visitor scenarios can be modeled by the following story algebra expression (semicolon is used for denoting sequential steps, plus sign for at-least-once iteration, square brackets for optionality and box for expressing alternatives):

$$mb; ps; (ir \Box (tp; (tsn; [ir])^{+})) \tag{1}$$

Given a storyboard (view) specification, a scenario schema must be *compatible* with the given scene transitions, which means the following: Atoms of the

story algebra expression must match to authorized scenes of the storyboard. The defined scenarios must correspond to valid directed paths within the storyboard (view). The defined scenarios must start with the marked initial scene and finish at the defined (or default) end scene(s).

Expression (1) is compatible with the visitors' storyboard view (Fig. 3). Consistency of the input-output content declarations can also be checked along the possible playouts. Note the link *6* will not be available if the visitor is coming from link *7* (there is no navigated route to go back to).

3.6 Decomposition Into Mini-Stories

A scenario or story schema might contain semantically meaningful, reusable patterns of scene transition playouts, which can be combined with each other flexibly. Story algebra expressions, however, may be too complex and hard to handle by human modelers, and such semantical information remains hidden. A possible solution is to take the union of the relevant scenarios and decompose them into *mini stories* [15] (or, at least, extract some mini-stories from it).

A *mini-story* is a semantically meaningful, self-contained unit, which can be used flexibly in different scenarios, sometimes by possibly different actors.It can be modeled explicitly and translated as a reusable subprocess in the workflow model. Syntactic hints or heuristics can reveal possible mini-story candidates, but at the end the modeler has to explicitly define or verify them.

In our case, given the story algebra expression (1), candidate mini-stories can be recognized by maximal, non-atomic subexpressions with none of its non-trivial parts appearing elsewhere. Based on modeler decision taking into account semantics as well, we define the following two mini-stories, and substitute them in the story algebra expression (in a real case, with more scenarios, their reusability could be better verified): 1. *Select location from map*: $Slfm ::= mb; ps$ and 2. *Navigate along trip (with reporting issues)*: $Nat ::= (tsn; [ir])^+$. We keep referring to scenes *ir* and *tp* as atomic mini-stories. The resulting story algebra expression with the above mini-story substitutions of (1) becomes:

$$Slfm; (ir \square (tp; Nat)) \tag{2}$$

3.7 The Story-Based Translation Method

After the storyboard (viewed by an actor role, refined and normalized) and the desired story schemata (story algebra expresions) are given as above, with the mini-stories modeled, the workflow model in BPMN for each story schema can be elicitated the following, inductive way:

- Translate atomic mini-stories,
- Translate compound mini-stories based on their story algebra expressions (which are not translated yet),
- Compose the complex workflow based on the full story algebra expression.

Translation can be hierarchically carried over using structural recursion along the story algebra atoms and connectives, as displayed on Fig. 5. A choice for rule alternatives is proposed to be determined using pragma-like declarations given for the usage model, or stereotypes associated to story algebra elements or subexpressions. For example, compund mini-stories can be translated as subprocesses, or connected using the link event notation.

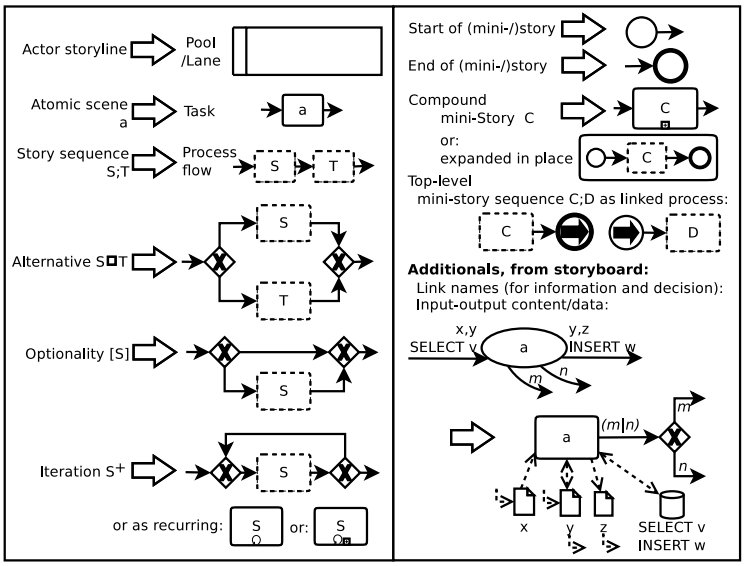

Fig. 5. Translation rules for story algebra expressions and additional assets based on storyboard. Dotted-lined rectangles denote arbitrary workflow model parts already translated from story subexpressions.

3.8 Enhancement of the Translated Model

Transition link names (here, numbers) can be added to the workflow model, as well as input-output content as data objects and database connections associated to the workflow tasks and subprocesses (see the additional rules of Fig. 5).

Fig. 6 displays a result of the refined, normalized visitors' storyboard view (Fig. 3) being translated to BPMN, based on story algebra expression (2) and mini-stories of Section 3.6, using rules of Fig. 5.

Model translation may be guided by additional information in forms of scene or link stereotypes.BPMN provides a variety of assets and some of them could be directly elicitated.Stereotypes offer more semantic information such as data or user-driven navigation, cancellation or rollback of started transactions, etc., to be mapped to native BPMN constructs.

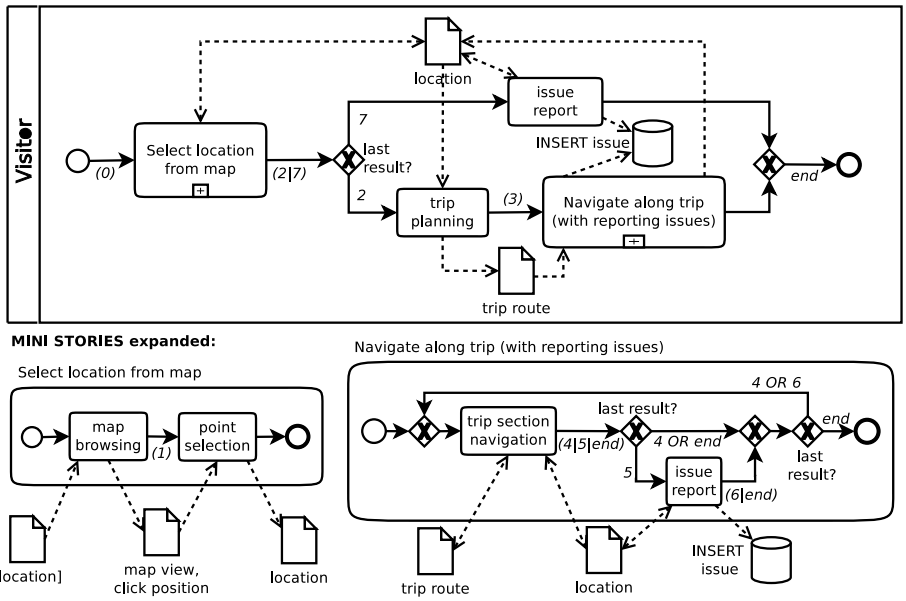

Fig. 6. BPMN workflow translation of visitors' view usage model, based on story algebra expression (2)

The modeling process is based on laying out default values for model formats, start/end scenes, authorized actor roles, context objects containing scenario history, handling of exceptions and invalid routing, stereotypes and other semantical or transformative guidance (e.g. how to connect mini-stories together, how to translate iteracted sub-processes). Defaults should work for conventional modeling cases. For customized, more sophisticated modeling, defaults can be overwritten. A possible post-translation optimization phase can improve the workflow model in each case. Most of these issues are left for future investigation.

4 Conclusion and Future Work

Model-based programming can be the true fifth generation programming, supported by sound foundation and appropriate tools, based on model suites of explicitly interrelated models. Models have their specific functions, viewpoints and can be given in various levels of details. Ensuring coherence, consistence and translatability among them is a crucial issue. In this paper, we have presented a general, layered modeling framework as a basis, and showed its feasibility by giving methods and guidelines for model development and translation between two specific types of models: the usage model (expressed by storyboard graphs and story algebra expressions) and the workflow model (expressed by BPMN).

The workflow model is claimed to be directly translatable to program code [10, 1]. We have introduced the concept of user view and the normalization of

the storyboard, providing guidelines to the modeler to refine an initial, top-level usage model. We gave two methods for translating the refined usage model to workflow models, and successfully applied the mini-story concept for semantically structured and flexible workflow elicitation.

The method is ready to be tested with more complex examples. Prototype implementations seem also possible. It opens up a variety of future issues, such as actor collaboration modeling, or defining stereotypes and pragmas determining model semantics and translations. The task of the metamodeler is to implement packages of generic models, and add-ons, which can enrich a generic model with pre-defined patterns and templates, and the actual application modeler can choose among them or let the modeling system decide on which defaults it uses for which cases. It points towards a generic model-suite framework, which is, in our view, essential for truly working general model-based programming.

References

1. The *Camunda BPM* manual. https://docs.camunda.org/manual/7.10/
2. Bienemann, A.: A generative approach to functionality of interactive information systems. Ph.D. thesis, CAU Kiel, Dept. of Computer Science (2008)
3. Chen, P.: Entity-relationship modeling: Historical events, future trends, and lessons learned. In: Software pioneers. pp. 296–310. Springer (2002)
4. Düsterhöft, A., Thalheim, B.: Information Modeling for Internet applications, chap. Systematic development of internet sites: Extending approaches of conceptual modeling, pp. 80–101. Idea Group Publishing (2003)
5. Knuth, D.E.: The METAFONTbook. Addison-Wesley (1986)
6. Lamport, L.: LaTeX: A document preparation system. Addison-Wesley (1994)
7. OMG: Business process model and notation (*BPMN*) version 2.0 (2010)
8. Pittman, T., Peters, J.: The Art of Compiler Design: Theory and Practice. Prentice Hall, Upper Saddle River (1992)
9. Schewe, K., Thalheim, B.: Design and Development of Web Information Systems. Springer (2019)
10. Stiehl, V.: Process-Driven Applications with BPMN. Springer (2014)
11. Thalheim, B.: Model suites for multi-layered database modelling. In: Information Modelling and Knowledge Bases XXI, volume 206 of Frontiers in Artificial Intelligence and Applications. pp. 116–134. IOS Press (2010)
12. Thalheim, B.: Normal models and their modelling matrix. In: Models: Concepts, Theory, Logic, Reasoning, and Semantics, Tributes. pp. 44–72. College Publications (2018)
13. Thalheim, B.: Entity-Relationship Modeling: Foundations of Database Technology. Springer (2000)
14. Thalheim, B., Jaakkola, H.: Models as programs: The envisioned and principal key to true fifth generation programming. In: 29th International Conference on Information Modelling and Knowledge Bases. IOS Press (2019)
15. Tropmann, M., Thalheim, B.: Mini story composition for generic workflows in support of disaster management. In: DEXA 2013. pp. 36–40. IEEE Computer Society (2013)
16. Türker, C., Gertz, M.: Semantic integrity support in *SQL:1999* and commercial (object-)relational database management systems. The VLDB Journal **10**(4), 241–269 (2001)