Innovations Syst Softw Eng (2009) 5:27–34
DOI 10.1007/s11334-009-0079-2

ORIGINAL PAPER

Bridging the requirements–implementation modeling gap with object–process methodology

Avi Soffer · Dov Dori

Received: 1 September 2008 / Accepted: 12 January 2009 / Published online: 10 February 2009

Abstract A model-based system development cycle involves two semantically distinct aspects: the requirements specification and the implementation model. Due to the conceptual and semantic differences between these two major system lifecycle stages, the transition from requirements to implementation is inherently a noncoherent process. Consequently, the system requirements are not faithfully transformed into the working system. This paper introduces an effective solution via an Integrated Modeling Paradigm (IMP) that combines the requirements and implementation domain models into a unified system model that continuously represents the system as it evolves. The IMP was implemented in an Object–Process Methodology (OPM) development environment. This implementation reinforces OPM with the capability to bridge the significant conceptual gap that lies right at the heart of the development process. A user survey has shown that this OPM-based solution is easy to use and can indeed help bridge the information gap, yielding a better match between the required and implemented systems than the currently accepted practice.

Keywords Model-driven development · Transition from specification to design

A. Soffer (✉)
Department of Software Engineering,
ORT Braude College, P. O. Box 78, 21982 Karmiel, Israel
e-mail: asoffer@braude.ac.il

D. Dori
Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology,
Technion City, 32000 Haifa, Israel
e-mail: dori@ie.technion.ac.il

1 Introduction

The complex nature of software-intensive systems introduces the challenge of narrowing the gap between the required system and its implementation. In spite of efforts to improve the flow of engineering information throughout the development process, oftentimes the implemented system does not fully match the required one, nor does it meet user needs and expectations [7].

The software development processes are supported by a variety of modeling approaches, based on the premise that modeling results in better systems [2]. However, due to the significant conceptual and semantic differences between the requirements and implementation models, islands of representation lead to abrupt, harmful transitions between these lifecycle stages [1]. Consequently, the system requirements are not faithfully transformed into the working system.

Maintaining a correct, clear, and coherent representation of the various aspects of the system under development can make a significant contribution towards successful system implementation and increased stakeholders' satisfaction. Bridging this requirements–implementation gap would enable coherent modeling and smooth transition between these development stages and prevent costly loss of information.

1.1 System development challenges

Developing systems in general and software systems in particular is a complex task that consumes significant time and expertise resources. Following the Model-Driven Engineering (MDE) paradigm, the development process has been steadily enriched with more powerful and more comprehensive modeling techniques.

Model-based system development reflects the system evolution throughout the different stages of the development cycle, including:

- Creation of conceptual models that represent the system in the specific stages
- Production of artifacts which are pertinent to each stage.

Maintaining a correct, coherent, and clear representation of all stakeholders' views along the development lifecycle has long been recognized as a major problem in the development of large, complex systems [6]. A fundamental challenge is to transform the system vision, as represented in the initial system's requirements model, into a working system that will realize that vision and respond to the stakeholders' needs [8].

Another challenge is the essential need to maintain the system's conceptual integrity. This challenge implies that capture and representation of the different stakeholders' views and needs be kept accurate and complete as the system's model evolves.

While the hope is that model-based development would improve the system quality throughout its lifecycle, this remedy comes with the price of adding complexity to the system development process.

1.2 The information discontinuity problem

A model-based system development cycle involves two major semantically distinct aspects: (1) the requirements and specification perspective, in which the system's needs are expressed and modeled in the problem domain, and (2) the implementation perspective, which includes synthesis of the system architecture and design, and the allocation of the required functionality to system components. The conceptual gap that separates these two domains of discourse is a significant source of difficulty and a major contributor to the undesired mismatch between the required system and its implementation. The problematic, sometimes incomplete, interpretation and translation of system requirements into a conceptual design, compounded with the difficult transition to the implementation phase, increases development time and probability of errors, raising the cost of the engineering process while lowering its quality.

A system is a collection of interrelated things that have static and dynamic aspects. In the world of software systems there are two types of contexts in which a system is modeled, one in the problem domain and the other in the solution domain. In the problem domain, the conceptual model of the system is typically known as the requirements/specification model. It models the system's needs and reflects the system requirements as elicited by the customer and analyzed for consistency and integrity by the developers. In the solution domain, the approach to implementing the required system is described by means of a selected system concept, architecture, and detailed design. Although these two perspectives, the problem domain and the solution domain, reflect aspects of the same system, they differ in a number of attributes, including the purpose they serve, the concepts they represent, the terminology they use, their intended audience, and their interaction with the relevant stakeholders.

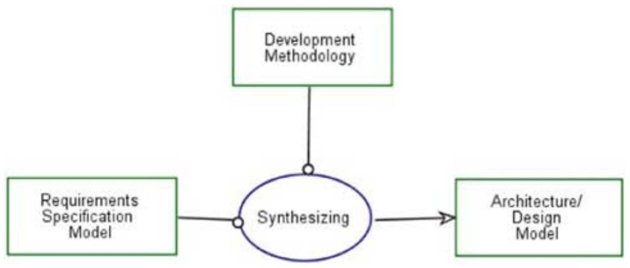


Fig. 1 Synthesis of implementation: the ellipse is a process that uses two sources as instruments and yields the architecture/design model

These differences cause the transitions between the requirements modeling phase and the architecting and design phase to be abrupt, possibly involving loss of information, knowledge, and intent. Disruptions in the flow of information are endemic; they occur regardless of the development methodology for two reasons: (1) there is inherent conceptual transition that is reflected in deep semantic differences, and (2) there are syntactic differences due to the different notations and interpretation of modeling.

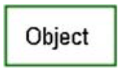
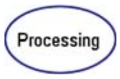
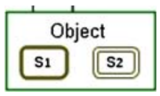
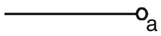
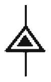
Transition from specification to implementation requires synthesis of the implemented system architecture in a way that enables performing the required functionality (Fig. 1). This mental transition is difficult and not well supported as part of the MDE paradigm [6].

Effective system modeling requires continuously maintaining the model integrity across the significant conceptual gap lying at the heart of the development process. Bridging the gap between specification and implementation will aid in this cognitive effort. This gap can be bridged by a continuous modeling framework, which facilitates coherent and consistent modeling of the system's specification and its implementation plan.

2 Object–process methodology

Object–Process Methodology (OPM) [4] is an emerging comprehensive modeling and development methodology, which strikes a unique balance between the system's structure and behavior [3]. Rather than requiring that the modeler views each of the system's aspects in isolation and struggles to mentally integrate the various views, OPM offers an approach that is orthogonal to customary object-oriented practices. According to the OPM paradigm, structure and behavior

Table 1 OPM building blocks

Element	Description	Symbol
Object	A thing that exist	
Process	A thing that transform objects	
State	An object can be exactly in one state in a specific point of time	
Procedural link	Connects entities to describe the behavior of a system	 [a]
Structural link	Expresses static relation between a pair of entities	 [a]

[a] A representative—the OPM lexicon includes more of this kind (see [4])

coexist in the same OPM model without highlighting one at the expense of suppressing the other to enhance the comprehension of the system as a whole. Complexity is managed via the ability to create and navigate via possibly multiple detail levels, which are generated and traversed by several abstraction/refinement mechanisms. Due to its structure–behavior integration, OPM provides a solid basis for representing and managing knowledge about complex systems, regardless of their domain.

The elements of the OPM ontology are entities (things: objects and processes, and states) and links. Table 1 presents a summary of OPM's core building blocks.

3 The integrated modeling paradigm

The integrated modeling paradigm (IMP) is proposed as an approach to narrowing the detrimental information gap between requirements and implementation. IMP combines the separate domain models along with the explicit and implicit information that associates these models to create an integrated system model. A key element of IMP is a new modeling component termed a *model connector*—an intermodel coupling mechanism that carries with it modeling information and enables information transfer between conceptually disparate domain models. In the IMP context, the model connector becomes an inherent part of the system model. It is created during the modeling process and embedded in the integrated model. Figure 2 is an abstraction of the integrated model including the embedded model connector.

The role of the IMP model connector is to represent the semantic connection between the domain models by linking applicable components to their counterparts in another model. The IMP model connector thus serves as a modeled bridge that fills the semantic gaps between the models.

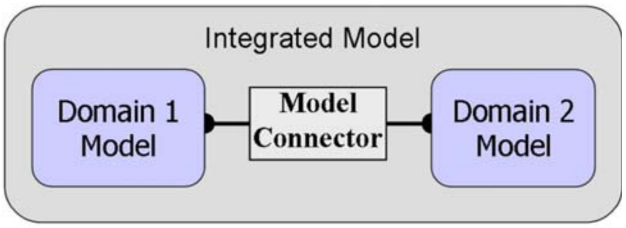


Fig. 2 An integrated model according to IMP

The IMP is intended to be used as an enhancement (i.e., an add-on) to existing modeling techniques. The model depicted in Fig. 3 reflects the addition of IMP as a characteristic of the existing development methodology. In order to realize the IMP vision, the development methodology should follow the prescribed model coupling methodology by using the IMP model connectors in the development lifecycle, leading to creation of an integrated model.

3.1 The OPM integrated modeling framework

Object–process methodology supports continuous system lifecycle modeling among other things. As described in Sect. 2, the theoretical foundation behind the OPM building blocks is minimalism of concepts and tight coupling of the static and dynamic aspects of a system, which results in a succinct set of symbols with precise semantics. This minimal concept set is combined with OPM's built-in abstraction/refinement mechanisms that enable gradual development of the system model, establishing a solid framework that supports the needs of integrated modeling.

As noted, the critical hurdle that hampers full realization of integrated modeling is the semantic gap between these two conceptually distinct models of the system—the requirements model and design model. To overcome this limitation we have extended OPM by adding a new construct named

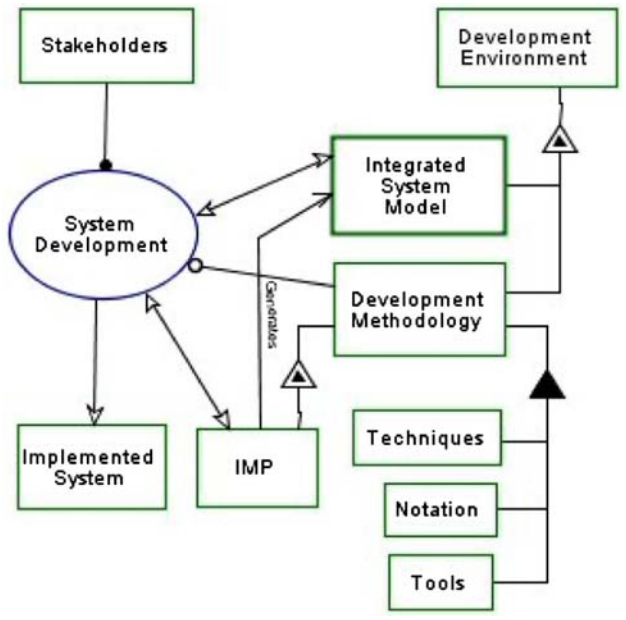


Fig. 3 Incorporating IMP into the development process

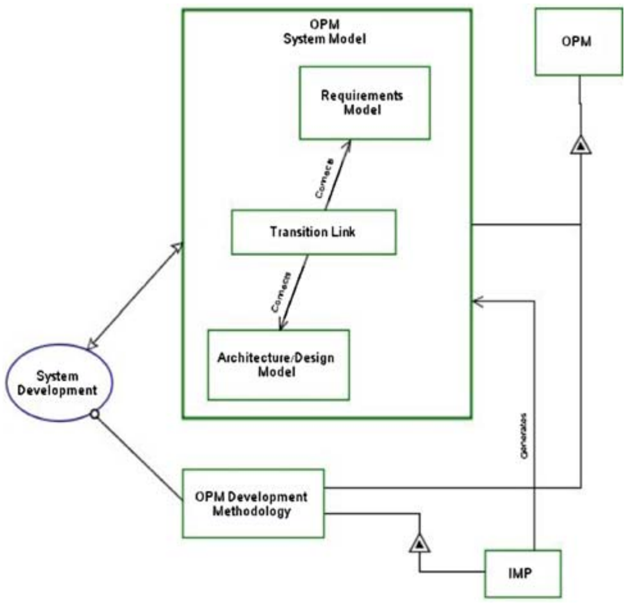


Fig. 4 Connecting requirements and design with the OPM transition link

a transition link. Figure 4 shows the role of the transition link in OPM integrated modeling. The transition link serves as a model connector that enables connecting OPM model elements in a way that supports integrated modeling of the requirements-design transition while bridging the semantic gap between them.

The transition link is constructed and added to the OPM model. To define this process we describe the specifics of the connection between requirements and design in the OPM environment in the Requirements–design linking metamodel, depicted in Fig. 5.

Based on this metamodel and following the generic create/bind/document steps that are part of the IMP approach, the process of linking things in our extension of OPM is as follows:

- **Creating**: the transition link is created. A unique identifier is assigned and registered in the *Identifier* field.
- **Linking**: The names of the things that are linked by the transition link are registered with the transition link in the two *Link End* fields. The things to be linked should have already been identified following the method of the transition between requirements to the design prescribed in Sect. 4—identifying, relating, and registering.
- **Documenting**: The engineering information and rationale for the design decision relevant to this link is captured and documented in the *Information* field.

From a methodology standpoint, creating the transition links and building the integrated system model are carried out during the transition from requirements to design as part of the iterative process of generating the architecture/design model.

One of OPM's fundamental principles is the abstraction/refinement capability that enables organizing the modeled system as a hierarchy of diagrams, called Object-Process Diagrams (OPDs) with the System Diagram (SD) being the top-level OPD. Using this mechanism, the integrated system model is represented by two subtrees, one representing the requirements model and the other—the design model. These two subtrees have a common root, which is one realization of the IMP.

This structure, materialized using OPCAT [5], the OPM modeling environment tool, is depicted in Fig. 6. Using OPM with our prescribed methodology, the OPM integrated model is initialized with the formation of two OPD tree hierarchies (one for the requirements model and one for the design model) with a common root. As the design of the system progresses, both the requirements and the design models and their corresponding OPD trees will be refined and expanded. Transition links will be created and used to link model elements in these two model sections, represented as the two major tree branches.

4 Coupling the requirements and implementation models

We demonstrate the OPM-based technical solution for integrated modeling via the home management system (HMS), a computerized system that provides comfortable living for

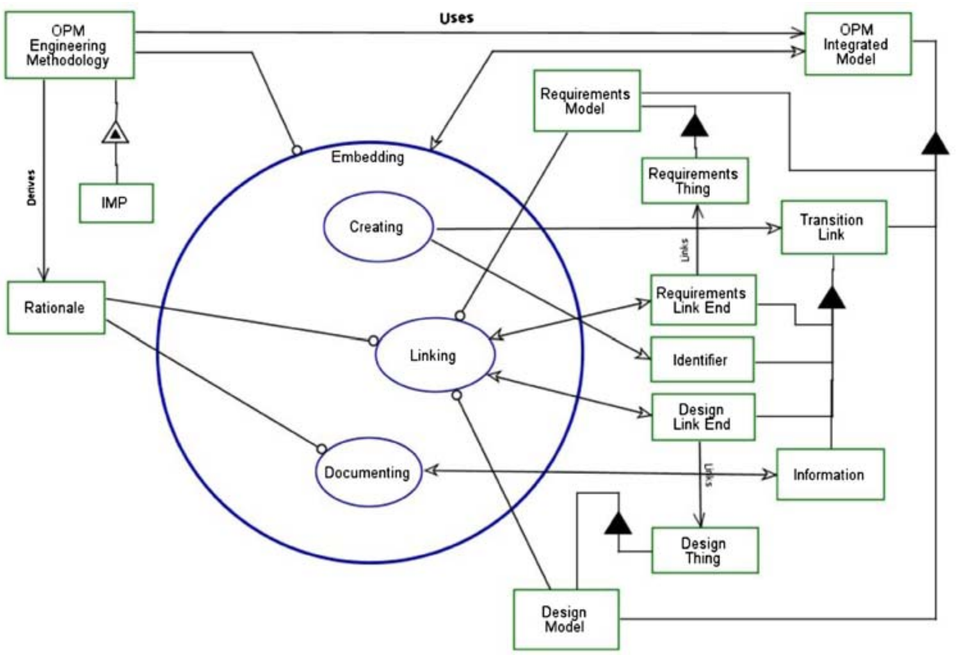


Fig. 5 The requirements–design linking metamodel

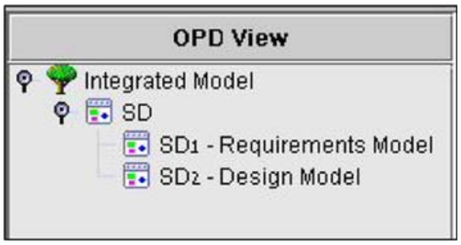


Fig. 6 The top of the integrated model OPD tree

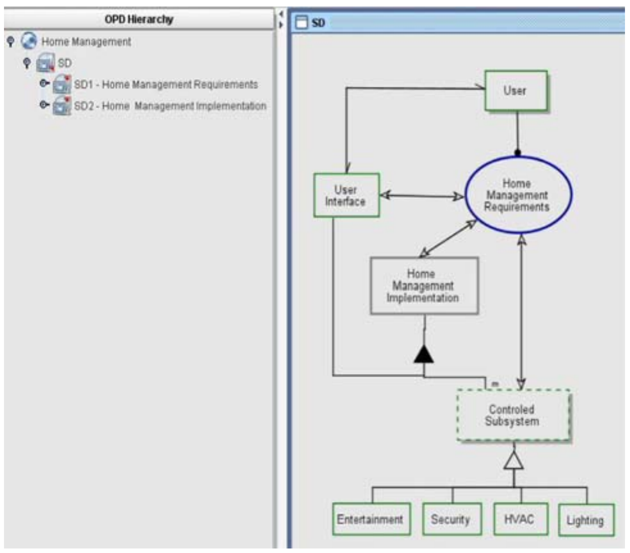

OPD Hierarchy
Home Management
SD
SD1 - Home Management Requirements
SD2 - Home Management Implementation
SD
User
User Interface
Home Management Requirements
Home Management Implementation
Controled Subsystem
Entertainment
Security
HVAC
Lighting

Fig. 7 Top-level system diagram of the home management system

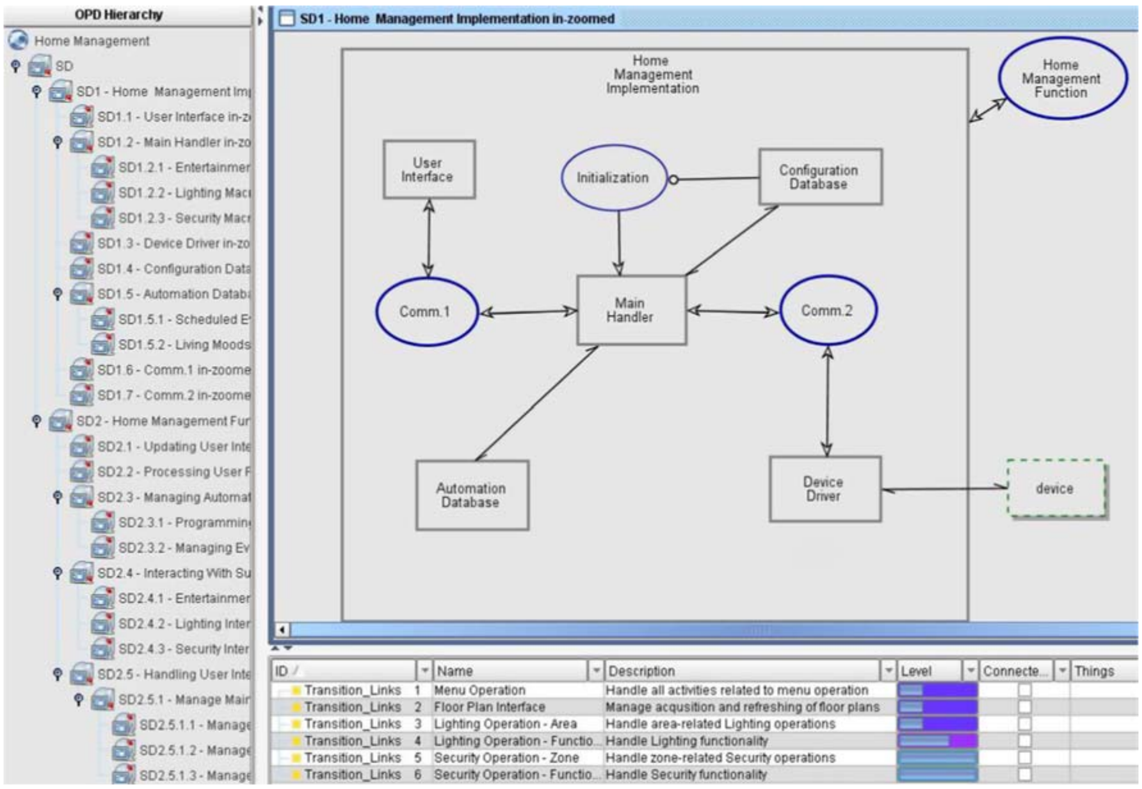
residents of large homes by easy control of the home living environment.

The HMS was modeled in OPM using OPCAT [5]. The top-level system diagram (SD) of "Home Management" is depicted in Fig. 7. The OPM integrated system model includes two main components, one representing the requirements model and the other—the design/implementation model.

The main OPD hierarchy view (on the left side) shows the top-level system diagram (SD), and immediately under it the roots of the two subtrees, SD1 and SD2, corresponding to the requirements and implementation models, respectively. As the modeling progresses, these two subtrees grow to reflect the incremental development of the domain models, as can be seen on the left side of Fig. 8. Some external entities such as the user, user interface devices, and the controlled subsystems are also shown in this top-level conceptual model of the system.

After establishing the requirements model and confirming it with the customer as needed, the design modeling was initiated. At the top of the design model is the HMS architecture, depicted in Fig. 8. This model includes several components which are needed to realize a system that fulfills the requirements, as described by the requirements model. Examples include the automation database, which records all the features programmed by the user and the initialization process, which performs all the automatic startup functions of the system.

The initial architecture model was then further refined to describe the more detailed design of the individual modules and their interactions. While the design progressed and the model became increasingly elaborate, transition links were constructed and added to the integrated model, gradually



ID		Name	Description	Level	Connecte...	Things
Transition_Links	1	Menu Operation	Handle all activities related to menu operation		☐	
Transition_Links	2	Floor Plan Interface	Manage acqusition and refreshing of floor plans		☐	
Transition_Links	3	Lighting Operation - Area	Handle area-related Lighting operations		☐	
Transition_Links	4	Lighting Operation - Functio...	Handle Lighting functionality		☐	
Transition_Links	5	Security Operation - Zone	Handle zone-related Security operations		☐	
Transition_Links	6	Security Operation - Functio...	Handle Security functionality		☐	

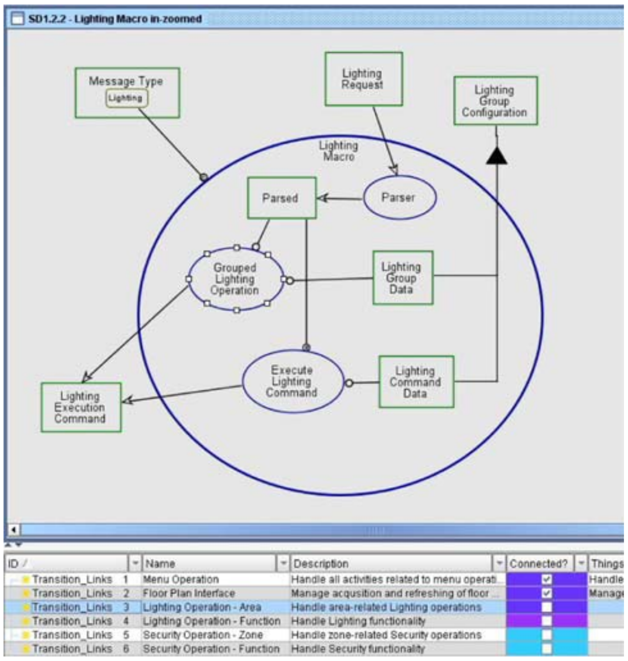
Fig. 8 The home management system's OPM architecture model

building the network of model connectors. As we developed the HMS model, we updated and deleted transition links as necessary.

We demonstrate the construction of the transition links via a particular example that shows how transition links are created and embedded in the OPM model. To avoid overloading the user with excess graphical information, the transition links that represent connections between two elements in different models are not visible in the model diagram. Instead, they are represented internally in a special data structure that includes the connection details and other information for each transition link. The transition links are arranged in a table named "Transition_Links", which is managed by OPCAT's configurable management view console, as shown below the OPD in Fig. 8. The user interacts with the links via this table.

The next step is binding the transition links to the proper OPM things (i.e., objects or processes) that need to be connected. The link binding process is shown in Fig. 9.

To bind the transition link (TL) to the corresponding model elements, we first select the TL of interest from the list (third line, bottom of Fig. 9). Then, we navigate through the OPD hierarchy view and select the OPD that includes the target model element that we want to connect, and select that thing (in this example it is the "Grouped Lighting Operation" process, as seen in Fig 9). To establish the link, we click on the "Connect" button. This creates a check mark in the "connected?" field. This operation can be reversed at any time by clicking the "Disconnect" button. To complete the binding, we repeat the linking process for the other side of the



ID		Name	Description	Connected?	Things
Transition_Links	1	Menu Operation	Handle all activities related to menu operati...	✓	Handle
Transition_Links	2	Floor Plan Interface	Manage acqusition and refreshing of floor ...	✓	Manage
Transition_Links	3	Lighting Operation - Area	Handle area-related Lighting operations	☐	
Transition_Links	4	Lighting Operation - Function	Handle Lighting functionality	☐	
Transition_Links	5	Security Operation - Zone	Handle zone-related Security operations	☐	
Transition_Links	6	Security Operation - Function	Handle Security functionality	☐	

Fig. 9 Binding a transition link to model elements

connection, namely “SD2.4.2 Lighting Interaction” in the requirements part of the model.

4.1 Findings

Using the IMP and OPM in the development of the home management system was illuminating in terms of understanding the strengths of the proposed methodology as well as in identifying some potential pitfalls. In what follows we describe these findings. The process of creating, binding, and documenting the transition links was very convenient and user-friendly. It was easy to navigate through the diagram hierarchy and quickly get to where we needed. The concurrent display of the diagram tree structure, the details of a selected component, and the link table in one view was instrumental in seeing the big picture, while working on the details of one particular part of the system. The automation provided by the tool, OPCAT (e.g., finding the connected things) was very helpful in increasing the productivity of the development process.

By working directly and concurrently with the two parts of the system model—the requirements model and the design model—we were able to explicitly relate the requirements to their implementation. This enabled keeping track of the progress of the design process and reflecting on our documented decisions when needed. While the process of creating the design from the requirements remained a complex and demanding task, the enhanced modeling capability served as a cognitive aid that alleviated some of the complexity that is inherent to this process.

The transition link was particularly useful in situations where the requirements component did not map directly to a single implementation counterpart. As one example, there might have been more than one target element on the implementation side to connect to, for a given requirement. As another example, a certain implementation component can be related to meeting more than a single requirement. In such cases the transition link made it clear that several implementation (or requirement) components are related to the same requirement (or implementation component).

We found that the enhanced modeling capability was very useful, but we also identified a few limitations. In particular, the fact that the requirements model does not lend itself to include nonfunctional requirements means that these had to be accounted for outside of the integrated modeling framework. Consequently, the transition links cannot be directly applied to nonfunctional requirements.

Nevertheless, we could document nonfunctional requirements in the transition link’s information field when these influenced a design decision. Another limitation is the way transition links are displayed. The list of transition links is flat and organized arbitrarily. When the number of transition links becomes very large, it could potentially be difficult to navigate through the list.

5 Conclusion

The requirements–implementation “Grand Canyon” is probably the most significant conceptual gap which lies at the heart of the development process. The proposed new bridging capability enables coherent modeling and smooth transition between these two key development stages. In this work we have described, demonstrated, and assessed the IMP—a solution for bridging the semantic gap between the requirements specification and the possible architecture, design, and implementation models without losing information captured in these models. Bridging this gap with IMP enables continuous modeling and explicit coupling of the requirements engineering activities and the rest of the development process, ultimately resulting in a better match between the required system and the implemented one.

Application of the IMP has been demonstrated via an intelligent home example, in which the unified model was built incrementally using transition links that provide detailed mapping between the requirements specification model and the architected implementation model. Analysis of the modeling example shows that an OPM-based system development process enables gradual and smooth transitions between development stages, enabling lifecycle-long coherent modeling while securing and maintaining the system’s conceptual integrity.

Applying a single modeling approach with transition links throughout the software development process is a paradigm that might greatly alleviate the inherent complexity of model-based system development. Based on the proof of concept presented here, this approach has the potential to ease difficulties related to requirements engineering, reuse, change control, verification and validation, and process management. This idea can be useful for any kind of system development process, as future research might be able to establish.

References

1. Boehm B, Port D (1999) Escaping the software tar pit: model clashes and how to avoid them. ACM Softw Eng Notes 24(1):36–48
2. Brooks F (1995) The mythical man-month. Addison-Wesley Longman, Reading
3. Dori D (1995) Object-process analysis: maintaining the balance between structure and behavior. J Logic Comput 5(2):227–249
4. Dori D (2002) Object-process methodology—a holistic systems paradigm. Springer, Berlin
5. Dori D, Reinhartz-Berger I, Sturm A (2003) Developing complex systems with object-process methodology using OPCAT. In: Int. Conf. on conceptual modeling (ER 2003), Lecture notes in computer science, vol 2813, pp 570–572

6. France R, Rumpe B (2007) Model-driven development of complex software: a research roadmap. In: Proceedings, future of software engineering (FOSE '07), pp 37–54
7. Gibbs W (1994) Software's Chronic Crisis, Scientific American, Sep. 1994, p 86
8. Harel D (2001) From play-in scenarios to code: an achievable dream. IEEE Comput 34(1):53–60