Controlled Modeling Environment using Flexibly-Formatted Spreadsheets

Hisashi MIYASHITA
Cybernet Systems (*) /
IBM Research – Tokyo, Japan
himi@meadowy.org

Hideki TAI
IBM Research – Tokyo, Japan
hidekit55@gmail.com

Shunichi AMANO
IBM Research – Tokyo, Japan
e35063@jp.ibm.com

ABSTRACT

As modeling in software and system development becomes increasingly prevalent, many engineers need to collaboratively develop models spanning many disciplines such as requirements management, system design, software, etc. However, integrating modeling languages for various disciplines is challenging, because UML and SysML are too complex for many engineers to understand. Therefore, in complicated engineering processes, engineers with different areas of expertise often find it difficult to access the same information in different domain-specific modeling environments.

Our approach to address this problem is to share and edit the models as task-oriented spreadsheets, using a unified model (in UML or SysML) and a unified user interface (in the spreadsheet program). The formats of the spreadsheets are optimized for various tasks while the target models remain in a unified modeling language. Since the transformation between the spreadsheets and the models is automated and transparent, users do not have to be skilled with the modeling languages to edit the spreadsheets.

Using our novel approach, we were able to reduce the errors and time, and also the difficulty for each task without providing specialized training for the engineers. A preliminary user study showed that, by applying the spreadsheet-based approach, we could reduce the number of errors with less time for typical systems engineering tasks.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; D.2.9 [**Software Engineering**]: Management—*Productivity*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

General Terms

Design, Management

Keywords

Model-driven Software and Systems Engineering, Model Transformation, Spreadsheets, UML, SysML

(*)...present address

ICSE '14, May 31 – June 7, 2014, Hyderabad, India
Copyright 14 ACM 978-1-4503-2756-5/14/05 ...$15.00.
http://dx.doi.org/10.1145/2568225.2568231

1. INTRODUCTION

Modeling in many areas such as software, systems, and enterprises is maturing [37, 30, 19] because the higher levels of description during development are important to share information and collaborate among engineers. For example, systems engineering must integrate many disciplines when developing large systems and many engineers with different areas of expertise have to collaborate with each other [37]. Typically this involves project management, requirements management, system design, and domain-specific tasks such as mechanical, software, and E/E (Electrical/-Electronic) engineering.

However, integrating modeling languages for various disciplines is not easy. Although UML [24] and SysML [25], an extension of UML for Systems Engineering, are regarded as the most promising languages to unify modeling information in a single format, many studies have reported that the complexity of UML is a serious concern [27, 23, 8, 28, 2]. Petre reported that the majority of the informants did not use UML partly due to the overhead of understanding the notation [27]. Moody reported that the visual notations of UML have many problems such as symbol redundancies and excesses, visual ineffectiveness, and graphic complexity [23]. Dobing *et al.* concluded that UML is useful for clients and users to participate in the development and reviews of artifacts, but many UML components, especially class and activity diagrams, are not well understood. They also concluded that UML should not be limited to software professionals, and a greater understanding of UML in building systems is needed throughout organizations [8].

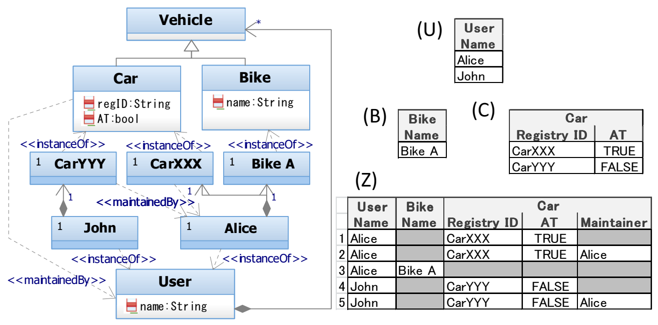
Although several existing studies addressed these problems by simplifying the UML metamodel [11, 31], altering the metamodel requires new modeling tools, new formats for models, and new model data transformations, all of which are very high barriers to delivering the new metamodels to the users. In our work, we address this issue by introducing a simplified presentation to edit the models while the underlying metamodels are unchanged.

We propose a novel approach to share and edit models using spreadsheets, our *SpreadSheet-based Modeling (SSM)*. It uses spreadsheets as simpler notations to create, edit, and browse the engineering information, and the formats of the spreadsheets are optimized for various tasks and disciplines while the target models are kept in a high-level UML or SysML representation. Since the transformations between the spreadsheets and the unified models are automated and transparent, users do not have to take special care to conform to the modeling languages when editing the spreadsheets with our novel modeling tool, *the Model Spreadsheet Platform (MSP)*.

Although there are a number of existing CASE (Computer-Aided Software Engineering) tools supporting import from or export to spreadsheets, or editing model elements in tables or matrices, such as Rational Software Architect, ReqPro, DOORS, and Rhapsody, to the best of our knowledge, none of them support for interactively editing both model elements and their relationships in tables by using spreadsheets (Section 2). Since this feature is vital for SSM, we illustrate it in Figure 1 with examples.

In the left panel, we show a model and a metamodel of our model. `CarYYY` and `CarXXX` are instances of `Car`, and `Bike A` is an instance of `Bike`. In addition, all of them are instances of `Vehicle`. `John` and `Alice` are instances of the `User` metaclass. Since the `User` metaclass owns the `Vehicle` metaclass, the instances of `User` owns the instances of `Vehicle` as shown in the diagram. All of the existing software can support limited tables as shown in (U), (B), and (C). In these tables, each instance has its own record (row) that shows the attributes of the instance. We cannot specify the relationships among vehicles and users in these tables, and we need distinct tables to show instances of each metaclass. Therefore, developing models by using the existing software is not practical.

MSP provides a more flexible spreadsheet format as shown in (Z) by combining (U), (B), and (C) to provide a holistic view. In this table, we can choose the second column or the third, forth and fifth columns to select `Bike` or `Car` metaclasses. If we change the name of `Alice` in Row 1 to `Alicia`, all the `Alice` cells in Lines 2, 3, and 5 are also automatically updated to `Alicia` in a just-in-time manner. If we remove Line 1, the `CarXXX` instance will be removed, and then Line 2 will also be removed. If we remove Line 2, the dependency of ≪`maintainedBy`≫ from `CarXXX` to `Alice` will be removed. Manually maintaining such tables is too cumbersome for users. Therefore, interactive editing with automatic updates is vital for modeling by spreadsheets. Even if we use flexible model transformations to import and export spreadsheets [4], we need to manually maintain such consistencies in a table, which is quite inefficient for practical modeling.



(U)

User
Name
Alice
John

(B)

Bike
Name
Bike A

(C)

Car	
Registry ID	AT
CarXXX	TRUE
CarYYY	FALSE

(Z)

	User	Bike	Car		
	Name	Name	Registry ID	AT	Maintainer
1	Alice	▨	CarXXX	TRUE	▨
2	Alice	▨	CarXXX	TRUE	Alice
3	Alice	Bike A	▨	▨	▨
4	John	▨	CarYYY	FALSE	▨
5	John	▨	CarYYY	FALSE	Alice

Figure 1: Examples of Models and Tables

We summarize the merits of our approach of flexibly-formatted spreadsheet modeling environment as follows:

Wide Acceptance

Spreadsheets are familiar to most computer users. Almost all of the project members, both non-technical people and engineers, understand how to use spreadsheets. This means we need less training for project members to use spreadsheets in comparison to specialized tools. Although quite a few modeling tools such as Rational Rhapsody and EMF [32] support tabular user interfaces, the configurable formats of these tools are too limited, and the user interfaces discourage many users. For example, we cannot cut and paste multiple cells in such tables even though such operations are standard in spreadsheets.

Flexible but Controlled Experience

Although UML provides various sorts of diagrams and a flexible extension mechanism to describe a vast range of concept for systems development, learning all of the language is quite difficult. Some researchers proposed adopting a critical subset of UML [2, 11], which could reduce the learning costs. This suggests that controlling the user experience while retaining flexibility is important for the practical use of UML. SSM allows us to choose an optimized representation for each model by using tables, matrices, or charts, which can be integrated into a single workbook to specify a wide range of systems development. Note that our basic approach is that the spreadsheet models coexist with the diagrams rather than replace them. We can choose the best method that matches each task, since MSP provides real-time synchronization between the spreadsheets and the models, as addressed in Sections 3.4 and 4.

Unified Model

Editing unified models with various spreadsheet representations is vital to manage and reuse the engineering information. MSP allows spreadsheet users to edit models based on a MOF (Meta-Object Facility) such as those supported by UML or SysML. This tool allows designing various spreadsheet formats and the users can edit the models without deep knowledge of the modeling languages. They only have to use the spreadsheet to edit and browse the model information. In addition, we can use other modeling tools that can access the unified models.

We applied our MSP to the actual development of an automotive transmission system that used four different types of representations. In Section 3, we describe the development method using SSM by showing the spreadsheet formats for the tasks. In Section 4, we give details about how we handled some selected tasks with the tool chain using MSP. Section 5 evaluates SSM with an experiment by executing the tasks covered in Section 3, and discusses the advantages and the disadvantages of our new approach. Finally, Section 6 provides our conclusions.

2. RELATED WORK

Our work is built on top of various technologies in the area of Model-Driven Architecture (MDA) in the sense that we use the unified modeling languages to describe engineering information, and propose a novel modeling tool that automatically offers a user-friendly experience using spreadsheets with support for collaboration. We classify these approaches based on three aspects: (1) Existing CASE tools supporting spreadsheets; (2) Automated modeling tools with adaptive or customized user interfaces; and (3) Model transformations for mapping models to tables.

Regarding (1), as stated in Section 1, we found that many existing modeling tools including Rhapsody have already introduced features to edit models in tables or matrices and to import and export spreadsheet files. However, to the best of our knowledge, none of the existing tools supports direct editing capability via these spreadsheets. While some tools such as Siemens Teamcenter® allow users to conveniently edit product information by automatically importing and exporting spreadsheets [29], users need to carefully edit the spreadsheets to avoid inconsistencies because the tools do not synchronize with the data managed by the tools before importing the spreadsheets. In addition, no tools support the editing of model elements and their relationships in an integrated table. MSP is the first attempt to implement a spreadsheet user interface as part of a modeling environment, and thus we were able to design a flexible spreadsheet format for editing the models, which is critical for the SSM approach.

In the area of spreadsheet user interfaces and modeling, Gencel [12], ClassSheets [10], and its successor, MDSheet [6] provide spreadsheet interfaces for UML models. First, these tools are different in scope from our work: they offer frameworks for model-driven software development for buisiness applications, while our MSP is intended for engineers who need to work with UML/SysML models. Second, while ClassSheets models can be transformed into UML models, they do not support the editing of existing UML models (though this is mentioned as future work in [10]). In contrast, MSP allows users to work with various types of models with a standard interface: Engineers can open, edit and save model files through the spreadsheets with MSP.

For Aspect (2), EMF (Eclipse Modeling Framework) [32] and GMF (Graphical Modeling Framework) [34] from the Eclipse Project are well known frameworks to support various DSM (Domain Specific Modeling) tools. Developers can design their own DSM with little effort by using them. Our MSP also uses EMF internally to manage the logical object model and to generally support DSMs. TIGER (Transformation-Based Generation of Modeling Environments) [9] and VMTS (Visual Modeling and Transformation System) [20] addressed the needs for flexible presentation frameworks to support multiple views for modeling. The work is related to ours in the sense that we want to provide an optimized interface for modeling. However, they do not support any spreadsheet interfaces.

For Aspect (3), MSP is technically classified as automated mapping between models and spreadsheets. For transparent editing, MSP also provides bidirectional transformations. Since model transformation is one of the core technologies in MDA, there have been many research projects in this area [7]. However, there are no studies that share our goals other than MDSheet and the related tools. Although many tools such as the Eclipse M2T Project [35] and Rational Publishing Engine [13] provide features to present models as tables by using templates, they only support one-way transformations and thus do not provide editing functions. Microsoft Office Excel Injector [4] provides a function to transform between Excel files as XML documents and models by using ATL [15], they also lack editing functions.

3. EXAMPLES OF SSM

In this section, we show examples of the spreadsheets and explain how to edit the models via the spreadsheets. Figure 2 shows the modeling process used in this example. In each pentagonal symbol, we show a name and a typical role for the task, because we assume that various engineering roles are required for this process. These tasks can be grouped into two phases. The upper phase is responsible for managing requirements, and the lower phase is for developing target systems satisfying the requirements. Since typically we need different types of people and information for these two phases, communicating consistenly in uniform information presentations throughout the process is challenging. First, in Section 3.1, we show an overall view of the models of this example. Next, in Sections 3.2 and 3.3, we explain how we can effectively use the spreadsheets to describe the models and provide controlled and efficient user experiences for each role to process each task.

3.1 Example Models

As mentioned in the introduction, we manage all of the engineering information in SysML, and each process accesses the unified repository to edit the information. We use a standard SysML editor (Rational Rhapsody) and MSP to compare the productivities of the two modeling approaches.

The target product was a transmission system for an automobile, which involves multiple disciplines spanning software design, mechanical engineering, and control systems. We assume many engineers in the various disciplines collaborate to perform these tasks. More specifically, the stakeholders are the sources of the requirements in Requirements Acquisition; the system analysts are responsible for analyzing the requirements and deriving the functional requirements; the software engineers are responsible for developing the software design of the system in Design Synthesis; and the customization engineers are responsible for customizing the system by tuning the parameters in Customization; and all of the engineers need to ensure the Traceability by managing the links among the artifacts during the various tasks.

Note that this example does not include state machine diagrams, sequence diagrams, activity diagrams, nor parametric diagrams [25], but MSP can also support such models. Since MSP regards models as directed graphs, and transforms such graphs into records by querying the model elements, it covers any types of models as directed graphs, which we will explain in Section 4. We had already applied these diagrams to MSP and SSM, mainly for supporting the Design Synthesis task, where we need to design the dynamic behaviors and constraints of systems.

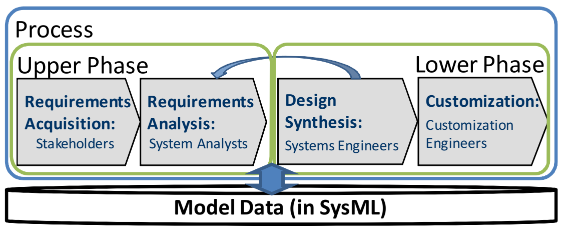

Process
Upper Phase
Lower Phase
Requirements Acquisition: Stakeholders
Requirements Analysis: System Analysts
Design Synthesis: Systems Engineers
Customization: Customization Engineers
Model Data (in SysML)

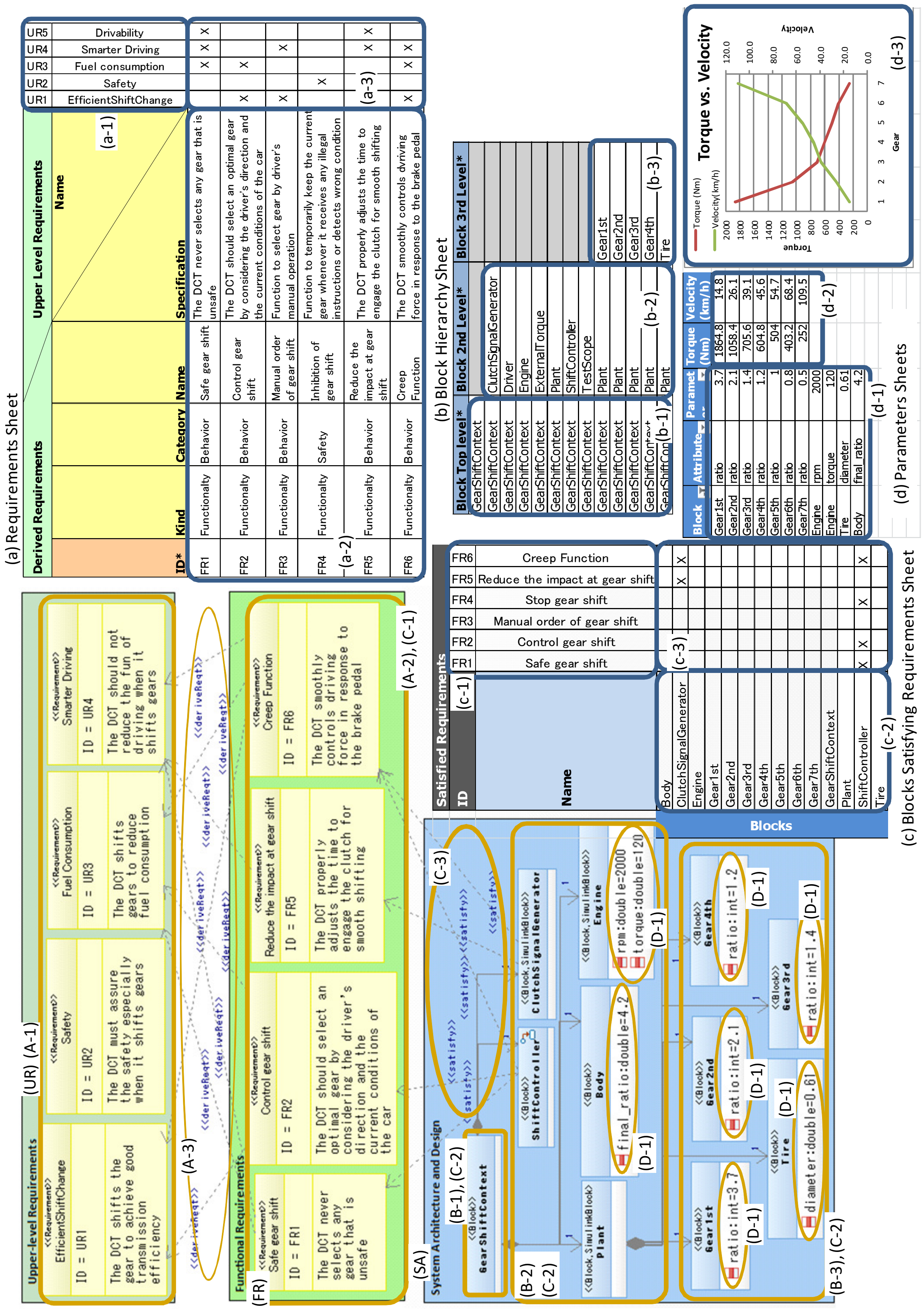
Figure 2: General form of the modeling process

In Figure 3, we show the models of this example as diagrams and spreadsheets. The parts denoted by (UR), (FR), and (SA) are SysML diagrams of the requirements and blocks. The parts denoted by (a-1)-(c-3) are spreadsheet representations. Notice that Part (X-n) of the diagrams corresponds to Part (x-n) of the spreadsheets, where X is capitalized and x is lowercase. For example, Part (A-1) corresponds to Part (a-1).

3.2 Requirements Modeling

In the upper phase, we manage all of the requirements that the target product must satisfy. In the step of Requirements Acquisition, we collect and list all of the original requirements coming from the stakeholders, which we call the upper-level requirements. The input is typically in the form of documents from the stakeholders. In the following step, Requirements Analysis, we identify all of the functions of the system by deriving the upper-level requirements, which we call the functional requirements.

This upper phase needs a comprehensive view of all of the requirements to browse and check them, since any incorrect or missing requirements in this phase will lead to project failure [3, 36]. For this purpose, we need a flexible tool to analyze and edit many items, with view-related functions such as sort and filter to check the consistency and the coverage of the full set of requirements. Tabular formats satify these needs and are widely accepted for requirement analysis. The SysML standard [25] defines an exception for a tabular format to present the requirements in other ways than a diagram format, and many requirements management tools such as Rational DOORS also provide tabular formats to edit requirements. In SSM, we exploit spreadsheet functions to achieve these aims. In Figure 3, Panels (UR) and (FR) show the upper-level requirements



(a) Requirements Sheet
(a-1)
(a-2)
(a-3)
(b) Block Hierarchy Sheet
(b-1)
(b-2)
(b-3)
(c) Blocks Satisfying Requirements Sheet
(c-1)
(c-2)
(c-3)
(d) Parameters Sheets
(d-1)
(d-2)
(d-3)
Torque vs. Velocity
Torque (Nm)
Velocity (km/h)
Torque
Velocity
Gear
Upper-level Requirements
(UR) (A-1)
<<Requirement>> EfficientShiftChange ID = UR1 The DCT shifts the gear to achieve good transmission efficiency
<<Requirement>> Safety ID = UR2 The DCT must assure the safety especially when it shifts gears
<<Requirement>> Fuel Consumption ID = UR3 The DCT shifts gears to reduce fuel consumption
<<Requirement>> Smarter Driving ID = UR4 The DCT should not reduce the fun of driving when it shifts gears
(A-3)
<<deriveReqt>>
Functional Requirements
(FR)
(A-2), (C-1)
<<Requirement>> Safe gear shift ID = FR1 The DCT never selects any gear that is unsafe
<<Requirement>> Control gear shift ID = FR2 The DCT should select an optimal gear by considering the driver's direction and the current conditions of the car
<<Requirement>> Reduce the impact at gear shift ID = FR5 The DCT properly adjusts the time to engage the clutch for smooth shifting
<<Requirement>> Creep Function ID = FR6 The DCT smoothly controls driving force in response to the brake pedal
(SA)
System Architecture and Design
(C-3)
<<satisfy>>
(B-1), (C-2)
<<Block>> GearShiftContext
(B-2)
(C-2)
<<Block>> ShiftController
<<Block, SimulinkBlock>> ClutchSignalGenerator
<<Block, SimulinkBlock>> Engine
rpm:double=2000
torque:double=120
(D-1)
<<Block>> Body
final_ratio:double=4.2
<<Block, SimulinkBlock>> Plant
(B-3), (C-2)
<<Block>> Gear1st
ratio:int=3.7
<<Block>> Gear2nd
ratio:int=2.1
<<Block>> Gear4th
ratio:int=1.2
<<Block>> Gear3rd
ratio:int=1.4
<<Block>> Tire
diameter:double=0.61

ID*	Kind	Category	Name	Specification	UR1 EfficientShiftChange	UR2 Safety	UR3 Fuel consumption	UR4 Smarter Driving	UR5 Drivability
FR1	Functionalty	Behavior	Safe gear shift	The DCT never selects any gear that is unsafe			X	X	X
FR2	Functionalty	Behavior	Control gear shift	The DCT should select an optimal gear by considering the driver's direction and the current conditions of the car	X		X		
FR3	Functionalty	Behavior	Manual order of gear shift	Function to select gear by driver's manual operation	X			X	
FR4	Functionalty	Safety	Inhibition of gear shift	Function to temporarily keep the current gear whenever it receives any illegal instructions or detects wrong condition		X			
FR5	Functionalty	Behavior	Reduce the impact at gear shift	The DCT properly adjusts the time to engage the clutch for smooth shifting				X	X
FR6	Functionalty	Behavior	Creep Function	The DCT smoothly controls dvriving force in response to the brake pedal	X		X	X	

Block Top level*	Block 2nd Level*	Block 3rd Level*
GearShiftContext	ClutchSignalGenerator	
GearShiftContext	Driver	
GearShiftContext	Engine	
GearShiftContext	ExternalTorque	
GearShiftContext	Plant	
GearShiftContext	ShiftController	
GearShiftContext	TestScope	
GearShiftContext	Plant	Gear1st
GearShiftContext	Plant	Gear2nd
GearShiftContext	Plant	Gear3rd
GearShiftContext	Plant	Gear4th
GearShiftContext	Plant	Tire

Satisfied Requirements

Blocks	FR1 Safe gear shift	FR2 Control gear shift	FR3 Manual order of gear shift	FR4 Stop gear shift	FR5 Reduce the impact at gear shift	FR6 Creep Function
Body					X	X
ClutchSignalGenerator						
Engine						
Gear1st						
Gear2nd						
Gear3rd						
Gear4th						
Gear5th						
Gear6th						
Gear7th						
GearShiftContext						
Plant						
ShiftController	X	X		X		X
Tire						

Block	Attribute	Parameter	Torque (Nm)	Velocity (km/h)
Gear1st	ratio	3.7	1864.8	14.8
Gear2nd	ratio	2.1	1058.4	26.1
Gear3rd	ratio	1.4	705.6	39.1
Gear4th	ratio	1.2	604.8	45.6
Gear5th	ratio	1	504	54.7
Gear6th	ratio	0.8	403.2	68.4
Gear7th	ratio	0.5	252	109.5
Engine	rpm	2000		
Engine	torque	120		
Tire	diameter	0.61		
Body	final_ratio	4.2		

Figure 3: The artifacts of our example in diagrams and spreadsheets

and functional requirements in the form of diagrams, and Panel (a) shows the corresponding requirements in the form of spreadsheets. (Due to space limitations, we intentionally removed some requirements from the diagrams.) In this example, MSP shows the upper-level requirements horizontally (a-1) and the functional requirements vertically (a-2), and displays the ≪derivedReq≫ dependencies in the matrix (a-3). At the same time, we can specify which attributes of each requirement should be displayed in the configuration of MSP, so we show IDs, Names, Specifications, and Kinds and Categories of the tag values of these requirements in this example.

In this spreadsheet, we can insert or update new columns to create or update upper-level requirements. Likewise, we can insert or update new rows to create or update functional requirements by using standard spreadsheet operations. By setting or clearing an 'X' value for cells in the matrix, we can create or delete ≪derivedReq≫ dependencies. MSP transparently handles almost all of the spreadsheet operations such as cut, paste, insert, and delete, and also provides an import function by inserting entries from external spreadsheets into the spreadsheet currently being edited, which are then reflected into the model. These functions are convenient to help input many entries into the models.

Operations that improperly change the layouts of the spreadsheets will be blocked by MSP with an error message. For example, moving the positions of tables or matrices, or adding or removing certain rows in horizontal tables (e.g. Table (a-1)), or adding or removing certain columns in vertical tables (e.g. Table (a-2)) will be blocked. This feature is important to guide users in correctly editing the models as spreadsheets, since users would frequently break the layouts of the spreadsheets without such protection.

3.3 Design Modeling

In the lower phase, we complete the design of the target system that satisfies all of the requirements defined in the upper phase. In the step of design synthesis, by considering the functional requirements, we create specifications for the component systems by deciding on their architectures and behaviors. This process requires sophisticated engineering expertise for covering the functions provided by each component. Since these decisions typically have major impacts on the products of the systems engineering [16], we need to carefully review and verify whether or not the results satisfy the requirements.

In contrast to the upper phase, software engineers have to deal with relatively more complicated structures and behaviors of the target systems such as architectures, activities, and states. However, especially in complex system designs, engineers often need to reuse existing assets and incrementally improve them. In typical cases, some templates should already be prepared and engineers will need to change the existing designs by following these templates. Therefore, we need a tool that can control the user experiences both for flexible and formulaic editing to fit each situation.

In SSM, we support flexible template design with the spreadsheets. In Figure 3, Panel (SA) defines the system architecture in the block definition diagram, and Panels (b) and (c) show the corresponding information in the spreadsheets. Panel (b) focuses on the hierarchy of the architecture, and Panel (c) focuses on the ≪satisfy≫ dependencies between the functional requirements and the architecture.

In Panel (b), the first column (b-1) lists the top level block, the GearShiftContext in (B-1), the second column (b-2) lists the second level blocks in (B-2), and the third column (b-3) lists the third level blocks in (B-3). By using Spreadsheet (b), the engineers can only create or delete blocks, or change the names of blocks, or change the hierarchy of blocks. They cannot change anything else in this sheet, such as editing the attributes or dependencies. If some spreadsheet templates predefine top level components, e.g. GearShiftContext, engineers can extend them by defining a new component hierarchy. Unlike diagram editing in which users can change any model elements, spreadsheet editing can effectively control the user's experience. Even if some diagram editing tools protect users from editing certain model elements, they will not be able to quickly understand what operations are allowed based on how they appear in the diagrams.

In Spreadsheet (c), Part (c-1) lists the functional requirements corresponding to Part (FR), Part (c-2) lists the blocks corresponding to Part (SA), and Matrix (c-3) shows the ≪satisfy≫ dependencies corresponding to Part (C-3). By using this spreadsheet, the software engineers can check whether or not all of the requirements are satisfied by the design by using a spreadsheet filter function as explained in Section 3.2.

For parameter tuning for customization, the benefits of the controlled user experience are large. Since the customization engineers do not need to change any design artifacts other than the parameters, the modeling tool insures that they focus only on editing the parameters. In our example, Part (d-1) in the spreadsheet (d) simply lists the parameters to be edited that are associated with the attributes of the blocks of Part (SA). Part (d-2) shows the calculated values from the parameters in (d-1) by using spreadsheet equations. Part (d-3) shows a graph of the values in (d-2). The customization engineers can concentrate on the parameters and the expected results by working on this spreadsheet. Spreadsheet modeling is quite suitable to provide these focused user experiences.

There are limitations in spreadsheet editing, since it does not present the shape of the structures of the model elements. For example, in Figure 3, the engineers can organize the software components, which are ShiftController and ClutchSignal-Generator, and place them close together to imply that they should be grouped. In contrast, in the spreadsheets (b) and (c), such groupings are not visible. In addition, users can attach more shapes and figures to the diagrams to convey their intentions. These features of the diagrams are beneficial for engineers to illustrate new concepts.

3.4 Synchronization of Models and Spreadsheets

Since our approach is designed to support synergy between the diagrams and spreadsheets as mentioned in Section 1, MSP can synchronize the changes between the spreadsheets and models. When users change the models by using Rhapsody, they can get the feedback in the spreadsheets, and vice versa. Therefore, when a system analyst changes some functional requirements in Sheet (a) in Figure 3 with MSP, the software engineers will receive the updates by using Rhapsody, allowing them to improve the design models to satisfy the updated requirements, since many modeling tools support diagram updates (which is called the populateDiagram feature in Rhapsody).

In a similar way, when users update the models via the diagrams, MSP automatically updates the spreadsheets as specified by the configuration. For example, if the systems engineers change the ≪satisfy≫ dependencies between the requirements and the blocks in Diagram (FR), the system analysts can see the updates in Sheet (c).

With this synchronization feature, MSP works well with the existing modeling environments such as Rhapsody, and this makes SSM applicable to the iterative processing between Design Synthesis and Requirements Analysis, as shown in Figure 2.

4. DESIGN AND IMPLEMENTATION

In this section, we describe the details of MSP that allow us to implement the examples described in Section 3. In brief, MSP can be viewed as a modeling environment on top of a spreadsheet that maps between the data of other models and spreadsheets.

4.1 Overview of MSP

Figure 4 shows the major components, inputs, and outputs of MSP. It uses a spreadsheet as a presentation component, i.e. MSP updates cells in spreadsheets to display the model data, and receives each user's input from the spreadsheets as change events. In the current implementation of MSP, we used Microsoft Excel™ [21] as the spreadsheet, and thus users see just Excel windows when editing models.

We call the spreadsheets controlled by MSP *"model spreadsheets"* and they allow us to edit model objects through the spreadsheet UI (User Interface). More specifically, MSP loads model objects from a Rhapsody model file via the Rhapsody API [14] when a model spreadsheet is opened, and presents the information on the spreadsheets. MSP updates the loaded model objects whenever the model spreadsheets are updated, and MSP saves the changed model objects in the target model file when the model spreadsheets are saved.

We are able to design the model spreadsheet by defining a configuration file in which we define the mappings between the logical tables and the model objects and also the layouts of the tables in the spreadsheet. MSP transforms the model objects into logical tables that are defined as collections of records, and then presents these logical tables in one of three types of layouts: a vertical table, a horizontal table, or a matrix. MSP uses the `ranges` of the spreadsheets feature to synchronize the data with the model, by creating a range for each table and matrix to catch any changes of the range. MSP ignores any other changes not in these ranges, and blocks any operations that try to move or delete these ranges.

MSP automatically maintains the consistency of the spreadsheet data. This is one of the major reasons why we need model spreadsheets for SSM, as stated in Section 1.

4.2 Configurations of MSP

In MSP, a configuration file specifies how to organize the vertical and horizontal tables and matrices by flexibly querying the model objects. The major feature for model spreadsheets is to update the spreadsheet information from the updates of the model objects without conflicts, and vice versa. In general, MSP regards the insertion and deletion of a record as the addition and deletion of a model element. We will clarify this mechanism by using an example, which corresponds to the example in Figure 1. We first show an example model in Figure 5 and an example configuration in Figure 6. In the following sections, we elaborate the features of MSP: (1) tree-based model-to-table transformation, (2) model editing via tables, and finally (3) the formal syntax and semantics of the configuration.

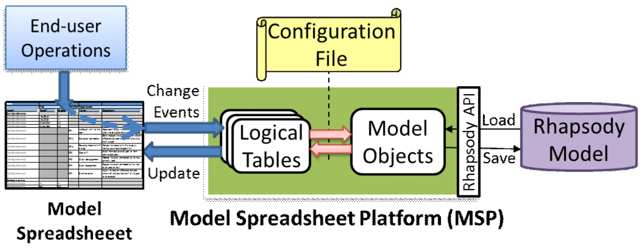


Figure 4: Arcthitecture Overview of MSP

4.2.1 Trees to Construct Records of Tables

MSP uses trees as intermediary data structures to transform model objects into records. In Figure 7, we show an intermediary tree and the corresponding table defined in Line 1 in Figure 6. The lines starting from `dim` in Figure 6 are called *dimensions*, by which we define the nodes of such trees, called *dnodes*, by associating them with the model elements specified in the path expressions. Line 2 defines `uDim` consisting of the `User` model elements in the `ownedElement`[1] of the root (the `Pkg` package), which are `Alice` and `John`. Each dimension may have an arbitrary number of columns. A dnode of Dimension `uDim` has a `uName` column by extracting the value of the `name` attribute from the model element associated with the dnode. As shown in the left panel of Figure 7, the two dnodes of Dimension `uDim` associated with `Alice` and `John` under Root were created because Line 1 specifies the `main` table starts from the `uDim` dimension.

Since Line 1 continues `(bDim | (cDim, dDim))`, the child dnode can be either `bDim` or `(cDim, dDim)`, which is called an *alternative* operation. This means that if the path of `bDim`, that is `./ownedElement[Bike]`, matches, `bDim` should be applied; otherwise, if the path of `cDim`, that is `./ownedElement[Car]`, matches, `cDim` should be applied, and then `dDim` will be applied if it is possible. Therefore, `bDim` is applied to the `Bike A` model element, and `cDim` is applied to the `CarXXX` and `CarYYY` model elements. Since `dDim` is applied only to `cDim`, the dnodes of Dimension `dDim` associated with `DepX` and `DepY` are the children of the dnodes of `CarXXX` and `CarYYY`, respectively.

Next MSP organizes a logical table by translating the path from the root to each dnode of the dimensions marked as `rec`, which we call `rec` dimensions, into a record, as denoted by the dotted lines in Figure 7. We do not create distinct records for `uDim` because it is not a `rec` dimension. MSP uses named perspective [1] to construct a record as a tuple. For example, the first record in Figure 7 is constructed as `<uName:Alice, regID: CarXXX, AT: TRUE>`, which comes from the pairs of the name and the value of the columns. The columns that do not have any values are called *empty* columns and are grayed out.

Then MSP lays out these logical tables as named tuples into horizontal (as shown in (a-1) and (c-1) of Figure 3) and vertical tables (in (a-2), (b), (c-2), and (d)), and matrices (in (a-3) and (c-3)). To present the records in vertical or horizontal tables, we specify the cell addresses associated with the column names of the record definition in an Excel template. For example, when we specify the column names as `uName`, `bName`, `regID`, `AT`, and `dName` in this order in the cell range of `B3:F3`, we obtain the vertical table (Z) in Figure 1. Likewise, when we specify these names in the cell range of `B3:B7`, we obtain a horizontal table by transposing the rows and columns of Table (Z). To show these tuples in a matrix, we combine vertical and horizontal tables with matrix cells by specifying which fields in a record are shown in horizontal and vertical tables of the matrix, and which field is shown in the matrix cells. For example, when we specify `uName` is presented in a vertical table, `bName` and `regID` are shown in a horizontal table, and `AT` is shown in matrix cells, we obtain the matrix shown in Figure 8.[2] In this way, by separating the logical table structures and layouts, MSP can flexibly map the model data into spreadsheets.

Regarding the scalability of MSP, each of the created records corresponds to a dnode of `rec` dimensions. Therefore, roughly speak-

[1]The UML specification defines `ownedElement` feature [24].
[2]All the key fields in the logical table must be specified in vertical and horizontal tables of the matrix to ensure more than one tuple is not assigned to one matrix cell.

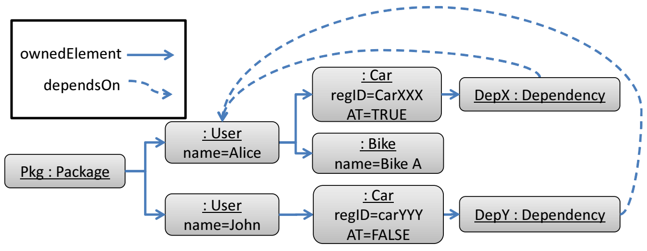


Figure 5: A sample model. `Pkg` is the root of this model.

```
table main = uDim, (bDim | (cDim, dDim))
dim uDim = /ownedElement[User], key uName=./name
dim bDim = rec ./ownedElement[Bike],                ↩
        ↪ key bName=./name
dim cDim = rec ./ownedElement[Car],                 ↩
        ↪ key regID=./regID, AT=./AT
dim dDim = rec ./ownedElement[Dependency],          ↩
        ↪ key dName=./dependsOn[User]/name
```

Figure 6: A sample configuration

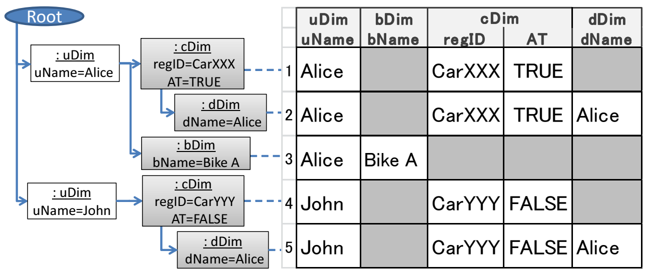
ing, the number of rows in the vertical tables is proportional to the number of dnodes of `rec` dimensions, and the sizes of the matrices are proportional to the squares of these dnodes. Because modern spreadsheet programs can support more than one million rows [22], we easily handled more than one hundred thousand model elements in MSP. Although this is adequate for practical use, MSP should support a model database or repository for large-scale model editing, which is future work.

4.2.2 *Editing Models via Tables*

MSP allows the users to edit the models via tables by translating (a) insertions, (b) deletions, and (c) updates of records into additions, removals, and updates of model elements, respectively. As shown in Figure 7, each dnode is associated with a model element. We illustrate each of these three operations.

Regarding (a) the insertion of a record, MSP first interprets it as the insertion of some dnode. MSP uses the key columns to identify which dnode is to be inserted. In Figure 6, `uName`, `bName`, `regID`, and `dName` are key columns, and each dimension must have at least one key column to identify the dnode from the values of the columns. By matching the key columns from the first dnode of the table, we can identify a path through the dnodes from the root. If the path fully matches an existing path, MSP reports an error to force the user to input the correct keys in the corresponding cells to avoid inserting a duplicate record in the table. Otherwise, MSP identifies a path through the dnodes to be added.

In Figure 9, we show an example of inserting records into the `main` table. In Case (I1), since `CarZZZ` does not match any record



	uDim	bDim	cDim		dDim
	uName	bName	regID	AT	dName
1	Alice		CarXXX	TRUE	
2	Alice		CarXXX	TRUE	Alice
3	Alice	Bike A			
4	John		CarYYY	FALSE	
5	John		CarYYY	FALSE	Alice

Figure 7: An example of `main` table (on the right) and an intermediate tree (on the left), where rectangles denote dnodes, and the dnodes of `rec` dimensions have gray backgrounds.

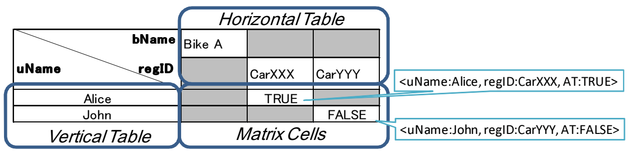
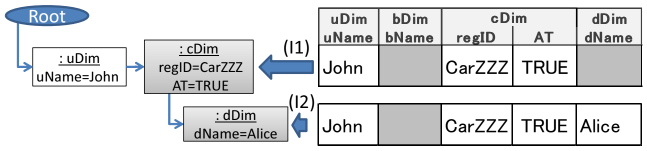


Figure 8: A sample matrix



uDim	bDim	cDim		dDim
uName	bName	regID	AT	dName
John		CarZZZ	TRUE	

John		CarZZZ	TRUE	Alice

Figure 9: Inserting records into the example table

in the table, a new dnode of Dimension `cDim` will be created. Note that users need to input the key columns of one of the dimensions in the alternative operation to identify it in the insertion record. Thus, in this case, the `bName` should not be input even if it is a key column. After that, by inserting row (I2), a new dnode of `dDim` will be created because `CarZZZ` matches the existing dnode. MSP allows adding only one dnode of the `rec` dimension in each record insertion, because all of the dnodes of the `rec` dimensions must have a one-to-one mapping with records. Thus, if we insert row (I2) before inserting row (I1), MSP will report an error.

After identifying the dnodes to be added, we need to translate them into the additions of the corresponding model elements. For this purpose, we deliberately designed the specification of a path expression to derive a model element satisfying a path expression (see Section 4.2.3 for details). For row (I1), MSP creates a `Car` model element to satisfy the path expressions of the `cDim`, and then sets the attributes of `regID` to `CarZZZ` and `AT` to `TRUE` based on the column definitions and the fields of the record. Finally, MSP adds this model element to the `ownedElement` of the model element associated with the parent dnode (which is the `John` model element).

When inserting row (I2), we need to add a model element corresponding to the dnode of the `dDim`. In this case, MSP creates a `Dependency` model element in the `ownedElement` of `CarZZZ`. In `dependsOn`, we can only set relationships and cannot add any model elements. Therefore, MSP tries to set the `dependsOn` relationship by searching for a model element of the type of `User` that matches the value of `name`, which is `Alice` in this case. If this fails, then MSP reports the error that it cannot find any reference targets.

Regarding (b) the deletion of a record, since each record has a corresponding dnode that has a corresponding model element, MSP simply removes the model element.

Regarding (c) the update of a record, in a similar way, MSP updates the corresponding model elements by considering the column definitions and the fields of the record.

As soon as the models are updated, Rhapsody checks the validity of the updates, e.g., equality and uniqueness of model element names, and if any of them are now invalid, MSP reports an error. If Rhapsody accepts these changes, then MSP updates the tables to reflect the changes. For this purpose, MSP uses the EMF notification mechanism [32] to receive the changes in the models.

4.2.3 *Syntax and Semantics of Configuration*

To define the syntax of the configuration of MSP, we assume countably infinite sets of the names of tables, dimensions, and columns, ranging over X, Y, and Z, respectively. Then the definitions of table and dimension are then defined as:

T	::=	X	(table name)
		Y	(dimension name)
		Y "," T	(sequence)
		T "|" T	(alternative)
D	::=	"rec"? P ("," C)*	(dimension)
C	::=	"key"? Z "=" P	(column),

where P is a path expression. The bindings of table and dimension names are given by a single, global set of definitions of the forms of `table X = T` and `dim Y = D`, respectively. For simplicity, in this grammar, we consider only the normal form of table definitions, which is defined as:

$$\begin{array}{lcl} \texttt{table}\ X & = & X_1 \,|\, ... \,|\, X_j \\ \texttt{table}\ X_1 & = & Y_{1,1}, ..., Y_{1,k_1} \\ ... & & \\ \texttt{table}\ X_j & = & Y_{j,1}, ..., Y_{j,k_j} \end{array}$$

To reduce the table definitions to the normal form, we assume the sequence, ",", and alternative, "|", operators are associative, and the sequence operators is distributive over the alternative operator, *i.e.*, $(T_1 \,|\, T_2)$, $T_3 = (T_1, T_3) \,|\, (T_2, T_3)$ and T_1, $(T_2 \,|\, T_3)$ $= (T_1, T_2) \,|\, (T_1, T_3)$. Then by considering Y is atomic in the algebra, we obtain the normal form by taking a sum-of-products [17]. For example, Line 1 in Figure 6 can be reduced to:

```
table main = X1 | X2
table X1 = uDim, bDim
table X2 = uDim, cDim, dDim
```

where `X1` and `X2` are temporary table names.

Before giving a formal definition of the path expression, we need to give a definition of a model in MSP as:

DEFINITION 1. *A model is a tuple $M = (W, S, N, V, E, f, n_0)$, where W and S are sets of type names and labels, N is the finite set of nodes of the model, V is a set of values, $E \subseteq (N \times S \times N) \cup (N \times S \times V)$ is the edge set, $f \in N \to W$ is a function to map nodes into type names, and $n_0 \in N$ is a root node. Each named edge $e \in E$ associates a node with a node or a value.*

The path expression is designed based on a subset of the notation of XPath [5], though we intentionally eliminated many complicated expressions that prevent editing models via tables. The formal syntax of a path expression is defined as:

P	::=	"/"	(root only)
		Pc	(otherwise)
Pc	::=	"."	(current)
		"/" A	(child 1)
		Pc"/" A	(child 2)
A	::=	S	(path element)
		S"[" Q "]"	(path element with qualifier)
Q	::=	Q "," Q	(sequence)
		W	(type name)
		S"="V	(equality expression)

The semantics of the path expression is given by the matching relation $n \vdash x \in P$, where o is a context node of the match. This relation is read "Value or node x is matched by P under the current node o," and the matching rules are defined as:

$$n_0 \in \texttt{"/"}\ (\textsc{Root}) \qquad o \vdash o \in \texttt{"."}\ (\textsc{Cur})$$

$$\frac{o \vdash \exists n \in P \quad (n, s, x) \in E}{o \vdash x \in P\&\texttt{"/"}s} \quad (\textsc{Child})$$

$$\frac{o \vdash n \in P \quad n \vdash Q}{o \vdash n \in P\texttt{"["}Q\texttt{"]"}} \quad (\textsc{Qualifier})$$

$$\frac{n \vdash Q_1\ ...\ n \vdash Q_k}{n \vdash Q_1\texttt{","}\ ...\ \texttt{","}Q_k} \quad (\textsc{Seq})$$

$$\frac{w = f(n)}{n \vdash w}\ (\textsc{Type}) \qquad \frac{n \vdash (n, s, v) \in E}{n \vdash s\ \texttt{"="}\ v}\ (\textsc{Eq}),$$

where P&"/" means it adds "/" to P only if P is not "/".
Notice that we can derive model elements from path expressions. For this purpose, by not discriminating between type names and types, we can define a function $g \in Q \to W$ to map qualifiers to a model element type as:

$$g(q) = \begin{cases} g(q_1) \cap g(q_2) & (\text{ if } q = (q_1, q_2)\ ) \quad (1) \\ q & (\text{ if } q \in W\ ) \quad (2) \\ \top & (\text{ otherwise }) \quad (3), \end{cases}$$

where $\top$ is the top type, and it is `EObject` in EMF. As shown in Equation (1), we assume that we can obtain the intersection of types. We also assume f^{-1} exists, which means we can instantiate a model element from a type name. If either of these operations fails, then MSP reports the error that it failed to create a model element of that type. Then the derivation rules from a path expression to model elements can be given as:

$$\texttt{"/"} \to \{n_0\}\ (\textsc{Inv-Root}) \qquad o \vdash \texttt{"."} \to \{o\}\ (\textsc{Inv-Cur})$$

$$\frac{o \vdash m \in P}{o \vdash P\&\texttt{"/"}s \to \{(m, s, f^{-1}(\top))\}} \quad (\textsc{Inv-Child1})$$

$$\frac{o \vdash m \in P \quad m \vdash Q \to E}{o \vdash P\&\texttt{"/"}s[Q] \to \{(m, s, f^{-1}(g(Q)))\} \cup E} \quad (\textsc{Inv-Child2})$$

$$\frac{n \vdash Q_1 \to E_1\ ...\ n \vdash Q_k \to E_k}{n \vdash Q_1\texttt{","}...\texttt{","}Q_k \to E_1 \cup ... \cup E_k} \quad (\textsc{Inv-Seq})$$

$$n \vdash s\ \texttt{"="}\ v \to \{(n, s, v)\} \qquad (\textsc{Inv-Eq})$$

With these rules, starting from path expressions, we can derive the nodes, edges, and values to be created in the model.

The semantics of a table definition can be given as model-to-tree relations, $n \vdash T \Rightarrow t$, where we assume the set of unranked and unordered trees, T_L, over dnodes, L, defined as $L = \{\phi\} \cup D \times N \times R \times ... \times R$, where ϕ is a root node of a tree and each $R = Z \times V$ is a pair of a column name and a value. This relation is read "The tree t is related to the table T under the dnode n." By using this relation, we can derive the tree t from the model M and the table definition T by $n_0 \vdash T \Rightarrow t$.

In order to derive these relations, we regard a table definition T as a set of trees, *i.e.* $T \in 2^{T_L}$, which is defined as:

$$\frac{D = \mathrm{Dim}(Y) \quad o \vdash l \in D \quad \mathrm{Node}(l) \vdash T = \{t_1, ..., t_k\}}{o \vdash l(t_1, ..., t_k) \in Y\texttt{","}T} \quad (\textsc{Child})$$

$$\frac{\begin{array}{c} D = (y, P, c_1, ..., c_k) \quad l = (D, n, r_1, ..., r_k) \\ o \vdash n \in P \ \ \forall i \in \{1, ..., k\}\ n \vdash c_i \Rightarrow r_i \end{array}}{o \vdash l \in D} \quad (\textsc{Dim})$$

$$\frac{c = (z, P) \quad r = (z, v) \quad o \vdash v \in P}{o \vdash c \Rightarrow r} \quad (\textsc{Column})$$

$$\frac{o \vdash t \in T_1}{o \vdash t \in T_1 \texttt{"|"} T_2} \ (\textsc{Alt-1}) \qquad \frac{o \vdash t \notin T_1 \quad o \vdash t \in T_2}{o \vdash t \in T_1 \texttt{"|"} T_2} \ (\textsc{Alt-2}),$$

where $\mathrm{Dim}(Y)$ is the dimension named Y, and $\mathrm{Node}(l)$ is n for $l = (D, n, r_1, ..., r_k)$. Then we derive a tree from a set of trees, T, by adding the root node, which is defined as:

$$\frac{o \vdash T = \{t_1, ..., t_k\}}{o \vdash T \Rightarrow \phi(t_1, ..., t_k)} \quad (\textsc{Root})$$

5. EVALUATION

We conducted a preliminary study by comparing the productivities of the approaches of SSM and General-Purpose-tool-based Modeling (GPM). Our objectives in this study are to evaluate (1) whether or not SSM allows a wider range of users to perform typical modeling tasks, and (2) what errors typical users made with SSM, compared to those made with GPM, to improve our approach and MSP.

5.1 Method

We chose seven participants from various disciplines, (1) a systems engineer, (2) a control system engineer, (3) a database engineer, (4) an application software engineer, (5) an embedded software engineer, (6) a verification engineer, and (7) a project manager, to cover a wide range of roles in practical engineering projects by following the research so that we would cover more roles rather than increase the number of participants [18]. Three of the participants (the systems engineer, the application software engineer, and the verification engineer) had more than five years of experience in SysML modeling, and the rest had less than three years of experience. All of the participants were familiar with Excel and Rhapsody.

Our research involved about two hours of work for each participant. First, we explained the diagrams and spreadsheets shown in Figure 3, and provided a short tutorial on editing models by using MSP and Rhapsody, which typically took thirty minutes, and then gave a task script for predefined tasks with MSP. After a fifteen-minute break, each participant performed the equivalent tasks with Rhapsody. This order was intentionally determined because we wanted to estimate the worst-case performance of SSM by giving a handicap to GPM for the task-learning effects. Finally, we conducted an interview to get detailed feedback.

We designed these tasks by reducing typical tasks in automotive development to reflect practical usage, since for such complex product development, we need to make small changes to reduce the impacts of each change. We used an automobile transmission SysML model consisting of about 100 blocks and 50 requirements, and asked each participant to perform four tasks. The first task was to create five requirements, each of which consists of an ID, Name, Specification, Kind, and Category. The second task was to create six requirements by deriving from the five requirements created in the first task by using the ≪derivedReq≫ dependencies. The third task was to edit seven SysML blocks by using compositions, and then create seven ≪satisfy≫ dependencies for the predefined requirements. The fourth task was to manage the traceability among the blocks and requirements by moving two dependencies and creating three new dependencies with the ≪satisfy≫ stereotype.

5.2 Results and Discussion

Figure 10 shows the results of the experiments. These results show that SSM is more efficient in both time and accuracy than GPM, although the number of errors the participants made varied. In our detailed analysis, the error ratios of the numbers of errors are 45.5% in SSM and 22.7% in GPM. By using the 95%-confidence intervals, the number of errors in SSM is 1.4 ± 1.3 while that in GPM is 5.4 ± 2.4. Regarding the working time, the error ratios are 11.8% in SSM and 6.9% in GPM. By using the 95%-confidence intervals, the time in SSM is 1,152 $\pm$ 271 seconds while that in GPM is 2,273 $\pm$ 313 seconds.

Our main interest in this study is to understand possible barriers for users to develop models in GPM and SSM. Figure 11 shows the analysis of the errors in both approaches. This indicates that the errors in SSM were only simple mistakes such as misspellings or incorrect names, or incorrect relationships, while the errors in GPM had a wider spectrum of errors, presumably due to the complexity of SysML. For example, in GPM, two users who had less experience in SysML modeling used ≪derive≫ instead of ≪deriveReqt≫. The other three users missed the ≪satisfy≫ stereotype. In UML/SysML, setting proper stereotypes seems to take a long time. Both the expert and novice users used reversed directions for dependencies, which also seems to confuse many UML/SysML users. Four users set incorrect requirement IDs and put data in incorrect slots because checking for the proper data in many different slots in SysML requires careful attention.

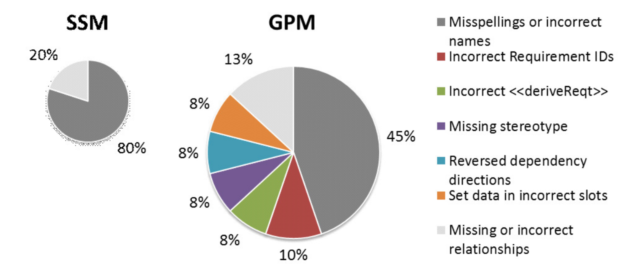
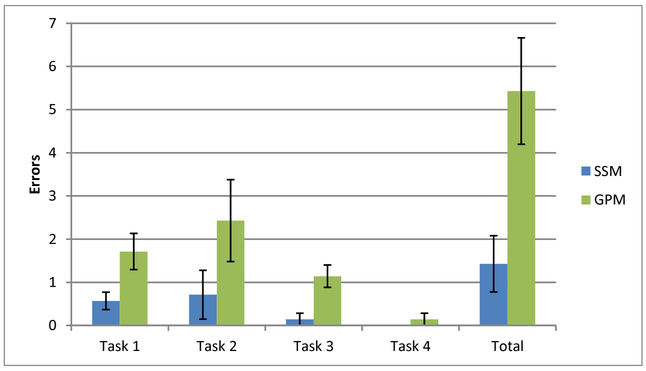


Figure 11: Analysis of the reasons for the errors. The sizes of the charts are proportion to the numbers of errors in SSM and GPM approaches, as shown in the left side of Figure 10

.

In contrast, these results show that SSM is more efficient, especially when the tasks can be clearly formulated. The configurations of MSP work as templates to guide users to edit the appropriate models. For example, since the dependencies between the upper level requirements and the functional requirements must be ≪derivedReq≫, this rule should be predefined in the configuration. Likewise, we should prepare spreadsheets to encourage users to fill ≪satisfy≫ dependencies for all of the functional requirements. As stated in Section 5.1, since we chose the typical and well formulated tasks that are executed by many engineers in actual automotive development, they fit with the SSM approach. This reflects our intention to apply modeling technologies to the area where many engineers in different disciplines work in established processes.

In the interviews with the participants, we received three major suggestions for improvements with SSM: (i) In SSM they were able to concentrate on editing the optimized spreadsheets, while in GPM they had to also consider the design of the diagrams and the details of SysML to achieve the goals; (ii) SSM provided the optimized spreadsheet formats to indicate the appropriate operations,



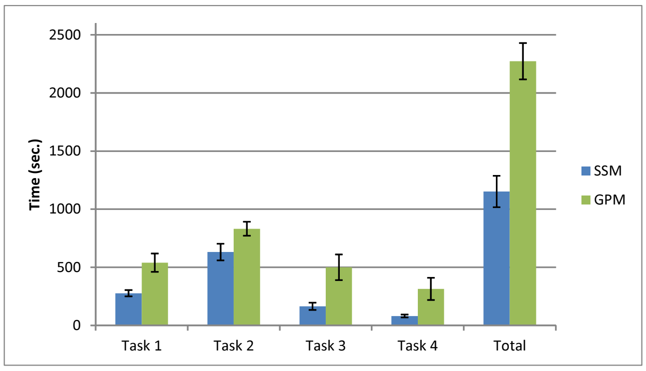


Figure 10: The number of errors (left) and the required time (right) on average for each task and total. The error bars show the standard errors.

while GPM could only provide some templates for the diagrams and these were not sufficient for the users to effectively understand the required operations; and (iii) To work with the complicated links (especially for the tasks involving Design Synthesis and Traceability Management), the users needed too much time to find the appropriate arrows, while in SSM they easily found these links by using the filters and sort functions.

These results and feedback implicated that the advantages of SSM came mainly from two factors: (a) the views were optimized for each task and (b) the efficiency of the spreadsheet UI. Separating these two factors leads to further research topics in modeling environments since (a) means that we should compare SSM with task-optimized user interfaces in the existing modeling tools; and (b) means that we should invent a generic spreadsheet-based modeling environment and then compare the productivity to the existing modeling tools. Although both of these areas have not yet been studied, both approaches are suggestive to improve modeling tools. We presume from our experiment that the errors shown in Figure 11 would be reduced by Factor (a) because task-specific views can reduce the complexity of UML/SysML, and the time for modeling is mainly reduced by Factor (b) because spreadsheets and Excel UI have been improved by many developers and are quite popular with many users compared to the existing modeling tools. For example, although Rhapsody provides advanced “Find/Replace” features to search for names, tag values, etc., users need to carefully specify which model information should be in the scope. In contrast, since MSP only shows model information in cells, users can easily use the search and replace features of Excel to edit models.

Regarding the efficiency of verification, both approaches have strengths and weaknesses. With GPM, we can check any kind of model element without any special configuration, while MSP requires some extended configuration data to handle some of the data as a spreadsheet. This implies that if the model contains unexpected errors, we cannot find some of these errors in SSM. However, in SSM, it is less likely that such bad data will be input compared to GPM. When checking for inconsistencies in the model information, MSP is better for browsing the large numbers of items. As the users reported in (iii), since the presentations with a general-purpose tool are less well organized, they require more effort to check the consistency compared to MSP.

With SSM, we need a configuration file for each task to customize the formats of the spreadsheets, while in GPM we need less work to prepare the templates of the diagrams. Thus, the configuration costs for SSM are larger than for GPM. However, when we need to deploy selected methodologies for a large number of team members, this overhead is less significant compared to the training overhead in GPM.

We conclude that the SSM approach fits well with development based on more established technologies, where all of the tasks and processes must be rigorously defined and thoroughly verified. In contrast, the GPM approach is suitable for research and experimental work, where many tasks are not defined in advance and depend on the situation.

6. CONCLUDING REMARKS

We proposed spreadsheet-based flexible modeling for developing systems, an application in which many experts in different disciplines have to collaborate with each other. We applied this approach to the four main tasks of systems development: (1) Requirements Acquisition, (2) Requirements Analysis, (3) Design Synthesis, and (4) Customization. We were able to design an optimized user experience on top of spreadsheets that enhanced productivity compared to generic UML/SysML Modeling tools. We did this by considering what information each user needs.

Our approach reduced the time and the errors for the typical modeling tasks in systems development. In our experiment, our approach was faster and more accurate than using general-purpose tools, In addition, we received feedback that they were able to perform the tasks with fewer errors.

We think this spreadsheet-based modeling approach has great potential to improve productivity in modeling by integrating various technologies such as Validation and Verification (V&V) and cloud-based scalable collaboration.

Regarding V&V, currently MSP itself supports only key constraints and Rhapsody checks the other constraints as discussed in Section 4.2.2. By integrating with a more general validation system such as EMF validation [33], we can specify a wider range of constraints for efficient model editing.

Regarding cloud collaboration for large scale model development, we are now continuing to work on incorporating OSLC (Open Services for Lifecycle Collaboration) [26] to support collaborative ALM (Application Lifecycle Management) and PLM (Product Lifecycle Management) to cover more processes beyond modeling throughout the development process.

7. ACKNOWLEDGMENTS

We greatly appreciate helpful discussions with Kohichi Ono, Yohichi Hatsutori, Ryo Kawahara, and Takeo Nakada.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*, volume 8. Addison-Wesley Reading, 1995.

[2] S. Ambler. *Agile modeling: effective practices for extreme programming and the unified process*. Wiley, 2002.

[3] T. E. Bell and T. A. Thayer. Software requirements: Are they really a problem? In *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, pages 61–68, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[4] H. Bruneliére. ATL transformation example, Microsoft Office Excel injector, 2005.

[5] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. URL: `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[6] J. Cunha, J. Fernandes, J. Mendes, and J. Saraiva. MD-Sheet: A framework for model-driven spreadsheet engineering. In *ICSE*, volume 12, pages 1412–1415, 2012.

[7] K. Czarnecki and S. Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17, 2003.

[8] B. Dobing and J. Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, 2006.

[9] K. Ehrig, C. Ermel, S. Hänsgen, and G. Taentzer. Generation of visual editors as Eclipse plug-ins. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 134–143, New York, NY, USA, 2005. ACM.

[10] G. Engels and M. Erwig. Classsheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 124–133. ACM, 2005.

[11] J. Erickson and K. Siau. Can UML be simplified? Practitioner use of UML in separate domains. In *proceedings EMMSAD*, pages 89–98. Citeseer, 2007.

[12] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Automatic generation and maintenance of correct spreadsheets. In *Proceedings of the 27th International Conference on Software Engineering*, pages 136–145. IEEE, 2005.

[13] IBM. Rational Publishing Engine. URL: `http://www-03.ibm.com/software/products/us/en/ratipublengi/`.

[14] IBM. Rational Rhapsody API reference. URL: `http://www.ibm.com/developerworks/wikis/display/Rhapsody/Rational+Rhapsody+API+Reference+Guide`.

[15] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1):31–39, 2008.

[16] A. Kossiakoff and W. Sweet. *Systems Engineering Principles and Practice*. Wiley Series in Systems Engineering and Management. John Wiley & Sons, 2002.

[17] R. Lennart, B. Westergren, et al. *Mathematics handbook for science and engineering*. Springer, 2004.

[18] G. Lindgaard and J. Chattratichart. Usability testing: What have we overlooked? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1415–1424. ACM, 2007.

[19] C. Marshall. *Enterprise modeling with UML: Designing successful software through business analysis*. Addison-Wesley Professional, 2000.

[20] T. Mészáros, G. Mezei, and T. Levendovszky. A flexible, declarative presentation framework for domain-specific modeling. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 309–312. ACM, 2008.

[21] Microsoft. Excel is a trademark of the microsoft group of companies.

[22] Microsoft. Excel Specifications and Limits. URL: `http://bit.ly/MnNSWn`.

[23] D. Moody. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, 35(6):756–779, 2009.

[24] Object Management Group. OMG Unified Modeling Language (OMG UML), v.2.0, 2005. URL: `http://www.omg.org/technology/documents/formal/uml.htm`.

[25] Object Management Group. OMG Systems Modeling Language (OMG SysML), v.1.3, 2012.

[26] OSLC. Open services for lifecycle collaboration. URL: `http://open-services.net/`.

[27] M. Petre. UML in practice. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 722–731. IEEE Press, 2013.

[28] K. Siau and Q. Cao. Unified Modeling Language: A complexity analysis. *Journal of Database Management (JDM)*, 12(1):26–34, 2001.

[29] Siemens. Teamcenter is a trademark or registered trademark of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in united state and in other countries. URL: `http://www.plm.automation.siemens.com/en_us/partners/extend_plm.shtml`.

[30] R. Soley et al. Model Driven Architecture. *OMG White Paper*, 308:308, 2000.

[31] J.-L. Sourrouille, M. Hindawi, L. Morel, R. Aubry, et al. Specifying consistent subsets of UML. *Promoting software modeling through active education*, pages 26–38, 2008.

[32] The Eclipse Foundation. Eclipse modeling framework. URL: `http://www.eclipse.org/emf/`.

[33] The Eclipse Foundation. EMF validation. URL: `https://www.eclipse.org/modeling/emf/?project=validation`.

[34] The Eclipse Foundation. Graphical modeling framework. URL: `http://www.eclipse.org/modeling/gmp/`.

[35] The Eclipse Foundation. Model To Text (M2T) - Eclipse. URL: `http://www.eclipse.org/modeling/m2t/`.

[36] A. van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pages 5–19, New York, NY, USA, 2000. ACM.

[37] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press. Elsevier Science, 2011.