INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED13
19-22 AUGUST 2013, SUNGKYUNKWAN UNIVERSITY, SEOUL, KOREA

SysML-BASED MODEL INTEGRATION FOR ONLINE COLLABORATIVE DESIGN OF MECHATRONIC SYSTEMS

Hongri FAN (1), Yusheng LIU (1), Ying LIU (2)
1: Zhejiang University, People's Republic of China; 2: National University of Singapore, Singapore

ABSTRACT

This paper introduces an online collaborative design platform to support mechatronic design. SysML-based system modeling method is employed to support system level design. Based on the system model, domain-specific model generation method is provided to facilitate the designers to enable the next design phase. The proposed unified recognition algorithm for changed model and dynamic model integration method enable efficient data flow to characterize the design intents in collaborative design activities. This platform is implemented on existing commercial tools to promote the practicability. Finally, a simple work piece conveyor system is taken as the case study for demonstration.

Keywords: online collaborative design, multi-domain, model integration, incremental update

Contact:
Hongri Fan
hrfan
State Key Lab of CAD&CG
Hangzhou
310058
People’s Republic of China
fanhongri@zjucadcg.cn

1 INTRODUCTION

The collaborative design has been recognized as a necessary development paradigm for complex mechatronic systems with the increasing globalization and the reduction of development cycle. It requires that the widely distributed designers should work seamlessly and timely to perform concurrent design tasks. Therefore, there is a great demand to enable the efficient design information flow across different design domains and phases. For the mechatronic product, it should be designed in an integrated fashion that the design modification in one domain should be propagated to other domains automatically to notify the engineers as soon as possible. Obviously, multi-domain model integration facilitates the mechatronic systems design in an integrated manner and the conflict solving in collaborative design. Usually, the gaps widely exist between different design tools, particularly across diverse domains. Moreover, as the rise of model-based systems engineering (MBSE), SysML-based system model promotes the significance of system design phase. As a result, the model integration across different design phases to speed up the design process becomes imperative as well. In summary, the issues for design integration of different domains and that of different phases mainly involves: (1) how to characterize the system model; (2) how to generate domain-specific model from the system model; (3) how to efficiently execute the dynamic model integration during the online collaborative design session.

In this paper, a metamodel-based model integration method is proposed to support online collaborative mechatronic design. Specifically, system design phase and detailed design phase for mechanical design domain and control design domain are involved. For these issues, the SysML-based system modeling method is first proposed to describe different aspects of the system structure. From the system model, metamodel-based mapping rules are defined for the model generation of different domains. And then the multi-domain model integration method is discussed. Finally, a uniform algorithm is given to recognize the dynamic change of different domain efficiently for incremental update. Please note that the focus of this paper is the execution of model integration rather than the determination for whether the model can be integrated or not. Therefore, the issues like concurrency and conflict detection are not involved here.

The rest of the paper is organized as follows. After reviewing the related works in section 2, the method is overviewed in section 3. The next four sections elaborate the system modeling, model generation, model integration during collaborative design activities and incremental update method separately. Implementation is provided in section 8 and the conclusion is given in section 9.

2 RELATED WORK

As mentioned in (Cabrera *et al*., 2008), an integrated modeling paradigm trying to reach a concept of the whole provided designers a proper view of the system, and the integration definition became possible as models can be represented in a common language. Qamar *et al*. (2010) described a multi-domain model integration framework to identify and solve dependencies across domains. It demonstrated that efficient design solutions and reduction of time are possible with concurrent multi-domain models integration. However, the critical model integration algorithm for information exchange was unimplemented. It was yet impossible to achieve model generation because of the lack of model transformation method. Chami *et al*. (2010) proposed an integration framework implemented by several linkage levels of SysML model with domain specific models, including requirement level, element level and attribute level for flexibility. A rough prototype with client/server structure was proposed to demonstrate the idea. However, as stated by the authors, lots of critical components like the communication protocol between server and client remained to be developed. Bajaj et al. (2011) introduced a collaborative, model-based integrated system platform－SLIM to federate domain-specific models such as simulation and optimization with the system model. It provides multiple types of model-connectivity patterns to connect the SysML-based system model to externally-defined design and analysis model. For instance, the CAD model parameters can be connected to the system model parameters at a fine-grained level. However, the connectivity would be invalid if CAD model parameters were not correspond exactly to the system model properties or if the topology of the CAD models is changed. The origin of the problem is the direct parameter relationship. Therefore, bi-directional model generation is critical to support the model integration for collaborative design between system-level design and detained design and that of different domains, and this point was also highlighted in (Qamar et al. 2010) and (Shahid, 2011).

3 METHOD OVERVIEW

The main idea of the method is to integrate the SysML-based system modeling platform with existing commercial CAD and control design platforms to provide a practical, useful solution for mechatronic system design. Meanwhile, the efficiency should be ensured to support online collaborative design. The overall structure of the proposed framework is shown in Figure 1. Usually, the design procedure starts with the system design until the system model is finalized. To facilitate the system design, SysML is employed due to its powerful extendibility to represent domain specific semantics. The extended profiles in this study involve the mechanical, control and kinematic domains to support characterizing different aspects of the system.

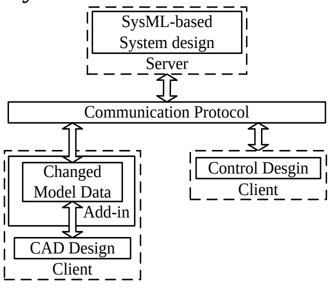


Figure 1 Overall structure of the platform

When the system design is finalized, the domain-specific design model should be generated by model transformation to enable next design phase. During the detailed design, the high-level system model acts as the server side to maintain the model mapping and transmission between involved domains while domain specific model stands for the client site. Collaborative session can be triggered by either server or client site. Client only contacts with the server to dispense the changed model data. For every site, as shown in CAD client, a plug-in resides in it to efficiently accomplish the collaborative design task such as recognizing, receiving and sending the changed model data. Moreover, to exchange model data between heterogeneous design systems, corresponding communication protocol is also necessary. To illustrate this platform, the MagicDraw™ is selected as the system design tool while CATIA™ and Matlab/Simulink™ for CAD design and control design respectively. However, the proposed method can be used for other platforms also.

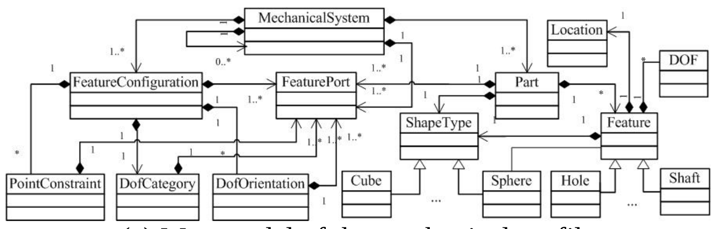
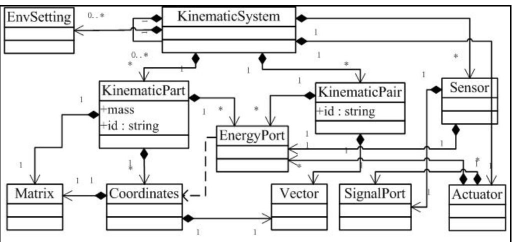
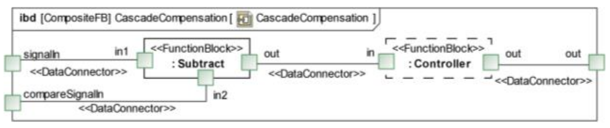
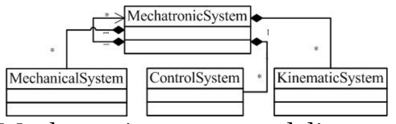
4 SYSTEM MODELING FOR MECHATRONIC DESIGN

During system design, designers need to model the system to get the abstract system structure which will direct the subsequent detailed design activities. Generally, the system structure model is finally represented by the high-level mechanical and control components. Here, for the control subsystem, two methods are available to represent the controlled plant. One is the causal modeling technology, which is implemented by employing a transfer function of the mechanical plant whereas the other one is the acausal modeling for system model which is widely used in uniform multi-domain simulation tools such as Matlab/Simscape, MapleSim and Dymola. The former method is less practical in system modeling as it's difficult to get the transfer function of the plant for designers. Furthermore, the incremental change propagation becomes impossible since the transfer function needs to be completely recalculated once the mechanical plant changes. Moreover, the latter facilitates the model integration with the simulation model and thus is recommended. According to the analysis, the profile should be defined based on the extension of SysML for modeling the mechanical, control and kinematics perspectives of the system. The adopted mechanical profile, as shown in Table 1(a), has been elaborated in (Fan *et al*., 2012) and thus is ignored here. Similarly, the control model profile is adapted from the stereotypes defined by (Cao *et al*., 2012). The core concept is Function Block(FB), which is defined in IEC 61499 to describe the high-level functional unit for control system. Taking the strategy "CascadeCompensation" shown in Table 1(b) as an example, it is modeled as a composite FB that includes the comparer FB and controller FB, connected by the data connector through data ports. The composite FB has an output data port which sends the signal to the controlled plant and an input data port to receive the output of controlled plant for comparing. The controller FB needs to be replaced by a concrete one like PID, which depends on the controlled plant. Based on these profiles, the mechatronic modeling profile is given to embody them as a whole. These profiles are defined by the "light-weight" method as it is easy to be implemented in the existing tools. In SysML, the *Block* is a modular unit of a system which can represent both logical and physical objects. Therefore, the stereotypes defined in this study are mainly extended from it for simplicity.

4.1 Kinematics Modeling Profile

The kinematics model is represented by the physical models and motion constraints between them to describe the physical structure rather than the mathematics equations. It enables the designers to model the mechanical system by connected block diagram. Moreover, it's very convenient to merge it into control design model to enable designer to design the mechanical and the control system in one common environment. Integrating it with the multi-domain simulation model like the SimMechanics and Modelica is also feasible as they share the same acausal modeling ideology, which result to similar meta-model mechanism. The related meta-models are defined as shown in Table 1(c). The main content of the meta-model includes two parts. The first part is the counterpart conceptions of the controlled physical part. The second is defined for online collaborative design. In the first part, the complete physical system is represented by the «*KinematicSystem*», which consists of the kinematic part for representing the rigid body and kinematic pair for joint. Kinematic part owns several coordinates, and the position is represented by a vector and orientation by a matrix. In addition, the matrix describes the inertia for the part while the vector represents the axis of the kinematic pair as well. The energy port through which the energy flows in/out of the part references the coordinates to indicate which anchor point it connects to. The second part involves the id information for part and pair, the usage will be elaborated later.

Table 1 System model Profile definitions

 MechanicalSystem Location DOF FeatureConfiguration FeaturePort Part ShapeType Feature PointConstraint DofCategory DofOrientation Cube Sphere Hole Shaft (a) Metamodel of the mechanical profile	 EnvSetting KinematicSystem KinematicPart +mass +id : string KinematicPair +id : string Sensor EnergyPort Matrix Coordinates Vector SignalPort Actuator (c) Kinematics modeling profile
 ibd [CompositeFB] CascadeCompensation [CascadeCompensation] signalIn in1 <<FunctionBlock>> : Subtract out in <<FunctionBlock>> : Controller out out <<DataConnector>> compareSignalIn in2 (b) CascadeCompensation composite FB structure	 MechatronicSystem MechanicalSystem ControlSystem KinematicSystem (d) Mechatronic system modeling profile

4.2 Mechatronic system modeling profile

Mechatronic system modeling needs to combine the multi-perspective models together to represent the system structure completely. For this purpose, «*MechatronicSystem*» stereotype is defined to represent the overall structure of the mechatronic system, which has mechanical, control and kinematics subsystems, as shown in Table 1(d). They are represented by the «*MechanicalSystem*», «*ControlSystem*», «*KinematicSystem*» stereotypes respectively. The control system includes the control strategy and specific controller. Such a profile enables the designer to freely configure the control strategy and the controller to generate alternative system structures.

5 DOMAIN-SPECIFIC MODEL GENERATION

For detailed design model generation from system design, meta-model based model mapping method is developed here. Actually, how to deal with the control model mapping is illustrated in (Cao *et al.*, 2012), and mapping for the kinematics model is implicitly provided in sec 4.1. Therefore, only the mapping between system model and detailed mechanical design model is explored here.

5.1 Detailed mechanical design model generation

Generally, the assembly model is composed of sub-assemblies, parts and the relationships between them. The relationship describes mechanical constraint between geometric features of parts. The metamodel proposed above is constructed by such an ideology as well. However, the feature configuration defined in system model has no counterpart in CAD systems, and thus should be mapped

based on the sub constraints. Table 2 shows the mapping rules from system model to CAD model. Here, CATIA is used as the exemplification. The "*ZeroDof*" stereotyped by «*DofCategory*» means the part is fixed, i.e. the "Fix" constraint coded as "*catCstTypeReference*" in CATIA API. For the feature configuration, it probably needs more CATIA constraint to get the assembly. Like the "*CenterTouch*", it is implemented by the combination of contact constraint and the coincidence of the center points for involved planes.
To generate the CAD model automatically, the part models are predefined and stored in database for the part primitives in system model, and the corresponding features are indexed as well. The detailed process can be referred in (Fan *et al*., 2012).

Table 2. Mapping rules between system model and CATIA model.

System Model		**CATIA**	
MechanicalSystem		CATIA_Assembly product	
Part		CATIA_Part product instance	
Feature	Hole	Hole	CATIA Feature
	Shaft	Pad	
DofCategory	ZeroDof	catCstTypeReference	CATIA Constraint
DofOrientation	RotMatch	catCstTypeParallelism	
	RotReversed	catCstTypeAngle(value=180°)	
	RotOriAngle	catCstTypeAngle	
PositionConstraint	Center-Aligned	catCstTypeOn	
Feature Configuration	CenterTouch	catCstTypeSurfContact, catCstTypeOn	
	HoleShaftMatch	catCstTypeParallelism, catCstTypeOn	
	PlaneDistance	catCstTypeDistance	
	Coplanarity	catCstTypeOn	

5.2 Coordinate system harmonization

It should be highlighted that generating the domain-specific models independently doesn't meet the requirement of multi-domain collaborative design. As the kinematic model needs the spatial information, like position from CAD model, it's recommended to have these models share the common coordinate system (CS). For example, the default gravity vector in SimMechanics environment is set to "*-Y*" direction which is easy to understand for control designer. Therefore, to make the gravity vector points to the same direction for generated CAD model, the "*ZeroDof*" constraint is employed for the reference part. This constraint removes all DOF to make the part keep stable. The desired CS is obtained if the reference part points to expected direction. Fortunately, this can be implemented by generating the "*catCstTypeReference*" assembly constraint and set the attribute to "*CATCstVal_Reference_Absolute*" for the part beforehand.

6 MAPPING BETWEEN SYSTEM MODEL AND DOMAIN-SPECIFIC MODEL

As mentioned before, the system model, CAD model and control model coevolve in online collaborative design activities. The essence of such design is how to relate these models together to allow efficient data flow between them. Generally, the issues involve: (1) During the transition from system design to detailed design phase, how to establish the connection between system model and the domain-specific models, i.e., the distributed counterparts; (2) How to find the changed model for the designed systems to achieve incremental update; (3) What information is needed to propagate the design intent, specifically from the control site to CAD site.

6.1 Domain-independent model identification

Generally, there are counterparts in other design systems for a given part. Generating the model connections for them is the prerequisite to support collaborative design. To build and maintain the relationship between them, the unique ID is used here. As the model changes freely in separated platform, it is provided for model indexing. This ID property accompanies the model at the background in every design system and keeps stable. It is initialized for every part in the system model and assigned to the domain-specific models during the generation. Obviously, the model needs to communicate with the outside requires the ID property. Every design system can find the counterpart exactly according to the ID index. In this way, the relation for all counterpart models is established.

6.2 Dynamic model generation

New models can be inserted arbitrarily when the design evolves, which necessitates adding the counterparts in other design systems. The model mapping across multi-domains for this case is achieved by two steps: (1) Mapping to the system model to reflect the new model at the high-level structure; (2) Based on the renewed system model, reusing the domain-specific model generation method to create the new model for the corresponding design systems. However, the direct mapping between domain-specific models is not preferred since the following reasons: (1) system model requires latest structure scene to relate it with functional and behavior models; (2) it removes the coupling between various domain-specific models since the system model actually acts as the interface to the other sites; (3) it induces the point-to-point topology structure at the model integration level.
As an example, the model mapping between the CATIA, system model and the SimMechanics model is defined in Table 3. It is easy to create the counterpart for the physical entity. For the CATIA joint, the basic ones can be mapped directly while some others need to combine several SimMechanics joints, like the CATIA CV joint, which needs two universal joints to represent it as the counterpart. How to represent the left complex CATIA joint by the composition of basic SimMechanics joints is under development. Please note that the ID property should be initialized from the source platform in dynamic model generation. For instance, if the CATIA site adds a new part, the ID should be set and then the others can maintain it to establish the model connection after receiving it. As the complex CATIA joint ultimately relates with more SimMechanics joints, a virtual counterpart is created at the background which includes the combined SimMechanics joints. As a result, the virtual counterpart holds the ID property and handles the communication task for inner joints.

Table 3. Multi-domain Model mapping

	CATIA	**System**	**SimMechanics**
Motion constraint	Revolute	Rotational	Revolute
	Prismatic	Translational	Prismatic
	Fix	Fix_Abs	Ground
	Gear	Gear	Gear constraint
	Screw	Screw	Screw
	Universal	Universal	Universal
	Spherical	Spherical	Spherical
	Rigid	Fix_Rel	Weld
	Cylindrical	Cylindrical	Cylindrical
	Planar	Planar	Planar
	Point Curve	PointAlongCurve	Point-Curve
	CV	CV	double Universal
	Slide Curve	SlideCurve	N/A
	Rack	Rack	N/A
	Cable	Cable	N/A
	Point Surface	PointOnSurface	N/A
Entity	CATIA_Part	KinematicPart	Body
	CATIA_Assembly	KinematicSystem	subsystem
	N/A	Acutator/Sensor	Acutator/Sensor(Body/Joint)

6.3 Qualitative relationship model

During collaborative design between detailed mechanical design and control design, it is easy to renew the related physical parameters for control design when they are changed in CAD systems. However, it is not a trivial task for the reverse change. If the request is to change the design parameter of a physical part, it can be directly reflected in the CAD model. Otherwise, the solution is undetermined. For example, supposing the request is to change the length of a part, which can be handled automatically as the length is a design parameter. However, if the request is to change its mass, it is impossible to decide which parameters should be changes since the mass is related to the volume and even the material. Therefore, a dependency model is proposed in this study to maintain and facilitate the change propagation between the control design domain and mechanical design domain, as shown in Figure 2. The dependency model includes the *result* property which represents the consequent parameters and the *cause* property stands for the cause parameters. With its help, the system model is able to determine which physical parameters should be adjusted conveniently; especially the change request comes from the control design domain.

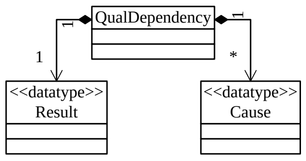


Figure 2.The Dependency model

Noticeably, although the relationship facilitates the system model to reason which parameter needs to change, it's not capable to cover all possible situations. When the CAD site receives the update notification for a part, some other statuses are possible: (1) the part still exists but the structure has changed which results to the relation model out of date in the system model. (2) The part has been removed which results to other counterparts become "dirty models". In such scenarios, an update should be triggered by the CAD site immediately to keep model consistency. The reason that these cases happen is mainly that there exist conflicts during cooperative multi-domain design, which is beyond the focus and left to be discussed in the future.

7 INCREMENTAL UPDATE METHOD

Incremental update from one design system to the others relies on two sub tasks: how to find the changed model from the local design system and by what communication protocol to distribute the information to other sites. For the first task, a unified algorithm is proposed to recognize the changed part, which can apply on different design platforms. Based on it, the communication protocol is defined for what should be sent out and how to specify the incremental semantics.

7.1 Unified recognization algorithm for changed model

As the change might happen on the arbitrary design system, the recognization algorithm for the changed model needs to be developed for all involved systems individually. However, to develop a unified algorithm, the model structure should be analyzed for involved design models. From the object oriented perspective, the design model structure consists of elementary model and relationships between them. In the system model, the part and feature configuration stands for the model and relationship in mechanical modeling profile respectively. The FB and the connector present them in control modeling profile; and the kinematic part and kinematic pair are the corresponding concepts in kinematics modeling profile. In the CAD model, the mechanical part and the motion joint implement these concepts. In the control model, the block and line represents them. Based on the above analysis, the core problem for the unified algorithm is how to find the changed parts and relationships. Supposing the model structure consists of M (models) and R (relationships) at time t_1: $M = \{m_1, m_2, ..., m_i\}$, $R = \{r_1, r_2, ..., r_j\}$ and $M' = \{m'_1, m'_2, ..., m'_g\}$, $R' = \{r'_1, r'_2, ..., r'_h\}$ at time t_2. Therefore, the contents of M and R at t_1 should be recorded first and then the algorithm can apply to find the changed models and relationships at t_2. The prerecording process is shown in the left of the Table 4. The attached *id* property is used for the domain-independent index across different design systems to support online collaborative design.

The main principle of the algorithm shown in Table 4 is constructing the model lists ML_1, ML_2 at time t_1 and t_2 first and then indicating the models in ML_1 which are not existed in ML_2 as removed. The new added models are recognized by indicating the models in ML_2 which are not existed in ML_1. For the left models in ML_2, checking the model properties changed or not. Therefore, all changed models can be found quickly. This algorithm can find the changed relationships as well. Applying it to specific design system requires it to provide the capability to get the pointers to all models, relationships and corresponding properties. Moreover, the model change criterion, i.e., properties comparison method should be implemented separately. Finally, the separated process for "New", "Changed" and "Removed" models depends on particular requirements and thus needs customization as well.

Table 4. Algorithm Description

Preparation of the algorithm:

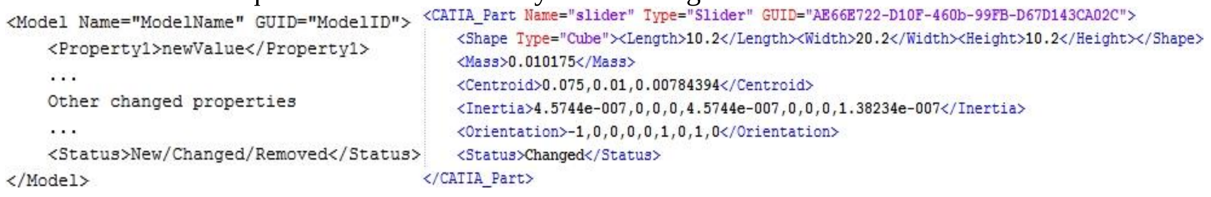
```
$prop_m$ : Model properties, including the
        detected properties for the model.
$prop_r$ : Relationship properties, including
        the detected properties for the relationship.
$id$ : A globally unique identifier.
$ptr_m$: The pointer of the model.
$ptr_r$: The pointer of the relationship.
$status$ : It indicates the change mode,
        i.e., "New", "Changed", "Removed", "Latest".
$s_m$ : A custom structure for model,
    $s_m = \{ptr_m, status, id, prop_m\}$.
$s_r$ : A custom structure for model,
    $s_r = \{ptr_r, status, id, prop_r\}$.
$removeModelList$ : A list for storing the removed model.
$removeRelList$ : A list for storing the removed relationship.
Begin:
Create a model list $ML$ and a relationshiplist $RL$.
For each model $m_k$ in $M$
    $s_m$ = new $s_m$(); $s_m.id$ = new $guid$(); $s_m.status$ = "Latest";
    $s_m.ptr_m = m_k.ptr$; $s_m.prop_m = m_k.prop$; $ML$.add($s_m$);
For each relationship $r_k$ in $R$
    $s_r$ = new $s_r$(); $s_r.id$ = new $guid$(); $s_r.status$ = "Latest";
    $s_r.ptr_r = r_k.ptr$; $s_r.prop_r = r_k.prop$; $RL$.add($s_r$);
End
```

Algorithm:

```
Begin:
// Construct a new pointer list for all models at $t_2$.
List $l$ = new List();
For each model $m_k$ in model structure.
    $l$.add($m_k$).
If $l$.empty() : // No model exists now.
    For each item $s_m$ in $ML$ :
        Set the status as "Removed" for $s_m$;
        Output id, and the status information for $s_m$;
        $removedModelList$.add($s_m$); // Store the resource for Recycling
    $ML$.clear();
    Return;
For each item $s_m$ in $ML$ :
    // Model is removed if it doesn't exist in $l$.
    If $s_m.p_m \notin l$ then $s_m.status$ = "Removed";
For each item $m_k$ in $l$ :
    // It's a new model if it doesn't exist in $ML$.
    If $m_k \notin ML$ :
        $\forall s_m$ in $ML$ & $s_m.status$ == "Removed";
        If $s_m \neq$ null: // Reuse a local resource in $ML$.
            $s_m.ptr_m = m_k.ptr$; $s_m.status$ = "Changed";
        else If ($removedModelList$.NotEmpty())
            // Reuse a global resource otherwise.
            $s_m$ = $removedModelList$.pop(); $s_m.ptr_m = m_k.ptr$;
            $s_m.status$ = "Changed"; $ML$.add($s_m$);
        else
            // Create a new $s_m$ if no resource available.
            $s_m$ = new $s_m$(); $s_m.id$ = new $guid$(); $s_m.status$ = "New";
            $s_m.ptr_m = m_k.ptr$; $s_m.prop_m = m_k.prop$; $ML$.add($s_m$);
    else if ($s_m.ptr_m = m_k.ptr$ & &
            !$m_k.prop$.equals($s_m.prop_m$))
        // The model exists in $s_m \in ML$, but properties change.
        $s_m.prop_m = m_k.prop$; $s_m.status$ = "Changed";
For each item $s_m$ in $ML$ :    // Process the changed model data.
    If ($s_m.status$ == "New")
        Process for new model;
        $s_m.status$ == "Latest";   // Reset status.
    If ($s_m.status$ == "Changed")
        Process for changed model;
        $s_m.status$ == "Latest";
    If ($s_m.status$ == "Removed")
        Process for removed model;
        $ML$.remove($s_m$);
        $removedModelList$.add($s_m$);
Release($l$);
End
```

7.2 Communication protocol definition

Essentially, the parametric dependencies across different design systems determine the communication contents in collaborative design. To reduce the transferred data to minimum, model change propagation should be based on the parameter rather than model itself. As mentioned above, the model changes in mechanical design must flow into the kinematics models to make the control design proceed successfully. Similarly, the parametric change for mechanical structure may be issued by the control design site. Therefore, the communication protocol can be divided into two parts: communication between system model and mechanical model; and that between system model and control model. The goal of the communication is incremental change propagation, which means only the changed parameters should be communicated with other design systems. According to such a principle, a changed-property-based pattern for the protocol content can be defined for the communication, as shown in Figure 3(a). It mainly includes the model name, ID and the status information to show which one is changed and what the changed mode is. The changed properties should specify the new values for "Changed" mode and all properties should be given for "New" mode. If the mode is "Removed", no property needs to be specified. Obviously, if the model is new added, the ID property will be stored by the receiver to relate the counterparts together. This uniform pattern applies to the change of model as well as the relationship concept mentioned above. According to the uniform pattern, an example for propagating the part model change from CAD model to system model is shown in Figure 3(b). It represents the communication content for a changed CATIA part model in CAD domain, all the changed information needed by control design is included and further used for updating the counterpart model on control design platform. From the control model to system model direction, it only includes the model ID and recommended parameter information. The status information is not required because it is always the "Changed" in such case.



```
<Model Name="ModelName" GUID="ModelID">
    <Property1>newValue</Property1>
    ...
    Other changed properties
    ...
    <Status>New/Changed/Removed</Status>
</Model>
```

```
<CATIA_Part Name="slider" Type="Slider" GUID="AE66E722-D10F-460b-99FB-D67D143CA02C">
    <Shape Type="Cube"><Length>10.2</Length><Width>20.2</Width><Height>10.2</Height></Shape>
    <Mass>0.010175</Mass>
    <Centroid>0.075,0.01,0.00784394</Centroid>
    <Inertia>4.5744e-007,0,0,0,4.5744e-007,0,0,0,1.38234e-007</Inertia>
    <Orientation>-1,0,0,0,0,1,0,1,0</Orientation>
    <Status>Changed</Status>
</CATIA_Part>
```

Figure 3. (a) Uniform communication style (b) Communication from CAD client to server

8 IMPLEMENTATION

To demonstrate the proposed method, a simple work piece distribution station is used as shown in the left of Figure 4. It consists of two mechatronic subsystems: a feeder unit (left) that provides the work piece and then the transfer unit (right) can transfer it to next station.

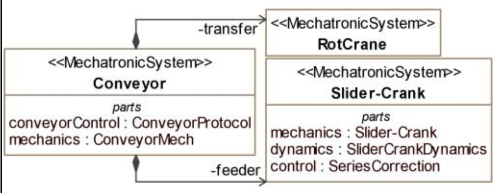
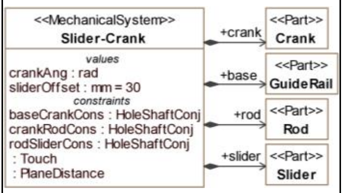
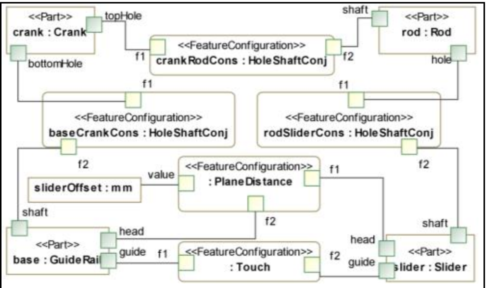
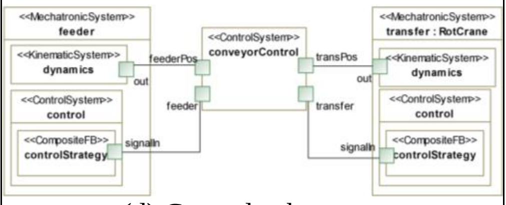
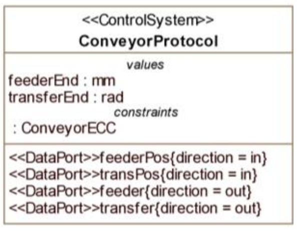
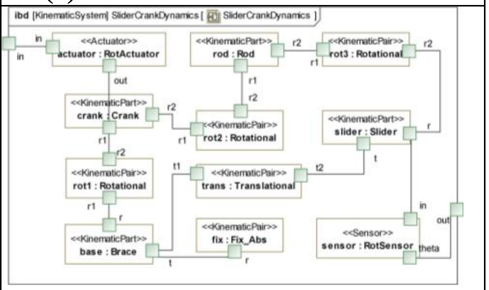
8.1 SysML-based system modeling

The MBSE tool Magic Draw 16.5 with SysML plugin is used to model the system design result. As shown in Table 5(a), the whole system is modeled by «*MechatronicSystem*» stereotype and named as "Conveyor", which has two subsystems. The conveyor has a logic controller which is defined by «*ControlSystem*» stereotype to coordinate two subsystems. Moreover, how to place the physical structure is represented by the mechanics property. For the kinematic models, they are embedded in subsystems like the "Slider-Crank" and no such model exists in the top conveyor system. Obviously, the whole system is modeled as a distributed design.

The mechanical structure of the feeder is defined as shown in Table 5 (b). The feeder is actually a Slider-Crank mechanism which includes the base, crank, rod and slider parts. These parts are connected by constraints. For example, the base is assembled with the crank by "*HoleShaftConj*" and the constraint is applied to assemble other parts, as shown in the Table 5 (c). The mechanical structure of the transfer is defined similarly and it can be assembled with the feeder similarly.

The two subsystems have the "CascadeCompensation" control structure as shown in Table 1(b). And the "ConveyorProtocol" type which tops them is defined to coordinate the logic sequence, as shown in Table 5 (e). According to detecting the state of the physical systems, it notifies the feeder when it should provide the work piece and when the transfer can remove it. Essentially, this task is performed by the "ConveyorECC" component, which acts as a finite state machine to model different system states driven by events, like the work piece ready event. The state transition is represented by the «*Transition*» stereotype between states. The central logic controller is connected with subsystem to coordinate the sub physical systems, as shown in Table 5 (d).

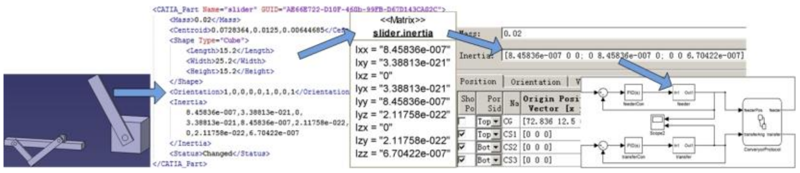
Table 5. System model for the case.

 -transfer <<MechatronicSystem>> RotCrane <<MechatronicSystem>> Conveyor parts conveyorControl : ConveyorProtocol mechanics : ConveyorMech <<MechatronicSystem>> Slider-Crank parts mechanics : Slider-Crank dynamics : SliderCrankDynamics control : SeriesCorrection -feeder (a) Conveyor system overall structure	 <<MechanicalSystem>> Slider-Crank values crankAng : rad sliderOffset : mm = 30 constraints baseCrankCons : HoleShaftConj crankRodCons : HoleShaftConj rodSliderCons : HoleShaftConj : Touch : PlaneDistance +crank <<Part>> Crank +base <<Part>> GuideRail +rod <<Part>> Rod +slider <<Part>> Slider (b) Mechanical definition of the feeder	 <<Part>> crank : Crank topHole bottomHole shaft <<Part>> rod : Rod hole <<FeatureConfiguration>> crankRodCons : HoleShaftConj <<FeatureConfiguration>> baseCrankCons : HoleShaftConj <<FeatureConfiguration>> rodSliderCons : HoleShaftConj sliderOffset : mm value <<FeatureConfiguration>> : PlaneDistance <<Part>> base : GuideRail head guide <<FeatureConfiguration>> : Touch <<Part>> slider : Slider f1 f2 (c) Internal structure of the feeder
 <<MechatronicSystem>> feeder <<KinematicSystem>> dynamics <<ControlSystem>> control <<CompositeFB>> controlStrategy feederPos out feeder signalIn <<ControlSystem>> conveyorControl transPos transfer <<MechatronicSystem>> transfer : RotCrane <<KinematicSystem>> dynamics <<ControlSystem>> control <<CompositeFB>> controlStrategy (d) Control subsystem	 <<ControlSystem>> ConveyorProtocol values feederEnd : mm transferEnd : rad constraints : ConveyorECC <<DataPort>>feederPos{direction = in} <<DataPort>>transPos{direction = in} <<DataPort>>feeder{direction = out} <<DataPort>>transfer{direction = out} (e) Central controller structure	 ibd [KinematicSystem] SliderCrankDynamics [SliderCrankDynamics] <<Actuator>> actuator : RotActuator <<KinematicPart>> rod : Rod <<KinematicPair>> rot3 : Rotational <<KinematicPart>> crank : Crank <<KinematicPair>> rot2 : Rotational <<KinematicPart>> slider : Slider <<KinematicPair>> rot1 : Rotational <<KinematicPair>> trans : Translational <<KinematicPart>> base : Brace <<KinematicPair>> fix : Fix_Abs <<Sensor>> sensor : RotSensor in out theta r1 r2 t1 t2 (f) Kinematic model of the feeder

The kinematics model of the feeder is shown in Table 5 (f), the mechanical part is represented by the «*KinematicPart*» stereotype and the joint by «*KinematicPair*» stereotype. The actuator and sensor are defined by «*Actuator*» and «*Sensor*» stereotypes respectively. As the model directly connected with control model, the corresponding actuator and sensor are added to merge them into control model.

8.2 Model generation and Incremental update

The CATIA model and the Simulink model are generated as the both ends shown in Figure 4. Control designer can add the *scope* block to observe the values. For the incremental update, supposing the slider instance has been replaced by another one, and then the recognization algorithm finds the changed model data as the XML format shown in the Figure 4. The part ID is reused from the removed part and the status is set to “Changed”, which notifies the other design sites to change the property directly on the old part, rather than repeats the removing and adding actions. The qualitative relation of physical parameters is not extracted since it doesn’t change. The system kinematics model updates when it receives the changed model data, like the inertia property. Finally, the SimMechanics model on the control design site is updated based on the system model to maintain the consistency. Actually, the related rod part as well as the corresponding joints of slider are also changed and propagated to the control sites, which are not shown explicitly. Moreover, some automatic processes are performed like the unit transformation and the minimal number is set to zero.



```
<CATIA_Part Name="slider" GUID="AE66E722-D10F-460b-99FB-D67D143CA02C">
    <Mass>0.02</Mass>
    <Centroid>0.0728364,0.0125,0.00644685</Ce
    <Shape Type="Cube">
        <Length>15.2</Length>
        <Width>25.2</Width>
        <Height>15.2</Height>
    </Shape>
    <Orientation>1,0,0,0,0,1,0,0,1</Orientation
    <Inertia>
        8.45836e-007,3.38813e-021,0,
        3.38813e-021,8.45836e-007,2.11758e-022,
        0,2.11758e-022,6.70422e-007
    </Inertia>
    <Status>Changed</Status>
</CATIA_Part>
```

```
<<Matrix>>
slider.inertia
Ixx = "8.45836e-007"
Ixy = "3.38813e-021"
Ixz = "0"
Iyx = "3.38813e-021"
Iyy = "8.45836e-007"
Iyz = "2.11758e-022"
Izx = "0"
Izy = "2.11758e-022"
Izz = "6.70422e-007"
```

```
Mass:    0.02
Inertia: [8.45836e-007 0 0; 0 8.45836e-007 0; 0 0 6.70422e-007]
```

Sho Po	Por Sid	Na	Origin Posi Vector [x
☐	Top	CG	[72.836 12.5
☑	Top	CS1	[0 0 0]
☑	Bot	CS2	[0 0 0]
☑	Bot	CS3	[0 0 0]

Figure 4. Generated CAD model, control model & the incremental update

9 CONCLUSIONS AND FUTURE WORK

In this study, a model integration based solution for online collaborative design platform is proposed. To support system design, it provides the SysML-based system modeling method to model different aspects for a mechatronic system. The metamodel-based model transformation method is provided to support the detailed model generation. As design involves, the incremental update is also supported according to the recognization algorithm and communication protocol. The advantages of the method include: (1) The multi-domain model generation as well as the model connection is provided to support the online collaborative design of mechatronic system. (2) Benefiting from the central hub of the system model, other domain-specific design tool can be integrated into this platform conveniently according to establishing the model mapping to the system model. (3) The uniform recognization algorithm for the changed model applies to various design systems and can be easily implemented. As a result, the size of data under transmission is very limited. (4) A prototype is implemented on existing design systems and thus users on different sites can still use familiar tools.

The system model mainly involves the layout design of the system structure, more aspects like the function and behavior model can be included to analyze the system design result at early design phase. In addition, the quantitative relation model can be developed to facilitate the parameter adjustment from control design site to CAD site. For the proposed platform, only three design systems have been integrated in this study, more systems need to be considered to enrich the platform in future.

REFERENCES

A. A. A. Cabrera, M. S. Erden, M. J. Foeken, and T. Tomiyama. (2008) ‘High level model integration for design of mechatronic systems’, *IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, Beijing, China, 2008, pp.387-392.

Qamar, A., Törngren, M., Wikander, Jan. and During, C. (2010) ‘Integrating Multi-Domain Models for the Design and Development of Mechatronic Systems’, *Proc. of 7th European systems engineering conf – EUSEC 2010*, Stockholm, Sweden.

Chami, M., Seemiiller, H., and Voos, H. (2010), ‘A SysML-based Integration Framework for the Engineering of Mechatronic Systems’, *Mechatronics and Embedded Systems and Applications (MESA), 2010 IEEE/ASME International Conference on*, Qingdao, ShanDong, 2011, pp.245-250.

Bajaj, M., Dirk, Z., Peak, R., Phung, A., Scott, A., Wilson, M. (2011a), ‘Satellites to supply chains, Energy to finance - SLIM for Model-Based Systems Engineering, Part 1: Motivation and Concept of SLIM.’, *21st Annual INCOSE International Symposium*, Denver, USA, 2011.

Shahid, H. (2011), ‘Integration of system-level design and mechanical design models in the development of mechanical systems’, Master Degree Thesis, Stockholm, KTH.

Cao, Y., Liu, YS., Fan, HR., Fan, B. (2012), ‘SysML-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics’, Computer-Aided Design (2012) doi:10.1016/j.cad.2012,05.001.

Fan HR., Liu, YS. (2012), ‘Integration of system-level design and detailed design models of mechatronic systems based on SysML and step ap 203 standard’, *Technical Presentation, Proceedings of the ASME 2012 IDETC/CIE, 2012, Chicago, IL, USA.*