Feature and Product Markup Languages in service-oriented CAX collaboration

A. Khaled

Electrical and Computer Engineering Department,
University of Alberta,
AB T6G 2G8, Canada

Y-S. Ma*

Department of Mechanical Engineering,
University of Alberta,
AB T6G 2G8, Canada
E-mail: yongsheng.ma@ualberta.ca
*Corresponding author

J. Miller

Electrical and Computer Engineering Department,
University of Alberta,
AB T6G 2G8, Canada

Abstract: The competitive and open market nature demands different vendors to collaborate during the product life cycle and to reduce the product's time to market. In this paper, we propose an infrastructure to enable the concurrent collaboration of heterogeneous Computer Aided tools for concurrent engineering aspect X (CAX) at the feature level using a Service Oriented Architecture (SOA) approach. A Feature Markup Language (FML) and a Product Markup Language (PML) are proposed as the modelling and communication media for feature and product information representation and exchanges which can be independent to operating system and programming language.

[Received 4 December 2008; Revised 9 July 2009; Accepted 20 July 2009]

Keywords: collaborative engineering; markup languages; PLM; product lifecycle management.

Reference to this paper should be made as follows: Khaled, A., Ma, Y-S. and Miller, J. (2010) 'Feature and Product Markup Languages in service-oriented CAX collaboration', *Int. J. Manufacturing Research*, Vol. 5, No. 1, pp.87–101.

Biographical notes: Adel Khaled is pursuing his PhD Degree in software engineering at University of Alberta. For the past four years, he has held the positions of technical manager and software architect at Netways, an e-Solution company based in Riyadh. He has over seven years of experience in software development with Microsoft .Net related technologies developing client-server windows based applications, web services and distributed applications.

He has implemented several projects using Microsoft technologies such as Content Management Server, SharePoint Portal Server, Microsoft Message Queuing, MS SQL Server and BizTalk.

Yong-sheng Ma joined the Department of Mechanical Engineering of University of Alberta since September, 2007. Before that, he had been with Nanyang Technological University in Singapore for seven years. His main research areas include product lifecycle management, feature-based product and process modelling. He received his BEng from Beijing TsingHua University (1986) and obtained both his MSc and PhD Degrees from UMIST, Manchester (1990 and 1994). In Singapore, he also lectured at Ngee Ann Polytechnic (1993–1996) and worked as a group manager and senior research fellow at the Singapore Institute of Manufacturing Technology (1996–2000).

James Miller has been with the Department of Electrical and Computer Engineering at the University of Alberta as a full Professor since 2000. His research interest is in software and systems engineering. He received the BSc and PhD Degrees in Computer Science from the University of Strathclyde, Scotland. He has investigated in the areas of software verification, validation and evaluation issues across various domains, including embedded, web-based and ubiquitous environments. He currently sits on the editorial board of the *Journal of Empirical Software Engineering*.

1 Introduction

Throughout the PLM, different processes and vendors are involved (Thimm et al., 2006). This mandates the use of different CAX systems that exist at disperse geographical locations with different ownership of the infrastructures. This raises several challenges, including but not limited to

1 difficulties of direct information sharing between the two tools due to format incompatibilities

2 the need to build custom components for information transformation and exchange

3 the loss of many of the semantics and design intents during the transformation and exchange process

4 software tools might be phased-out along the years

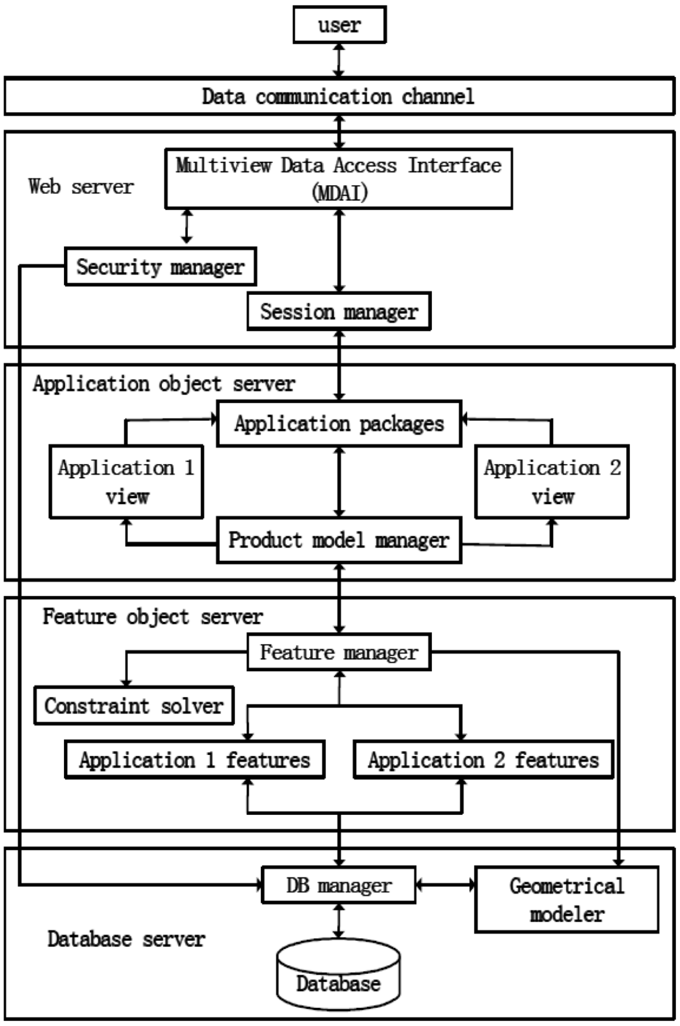
5 increased cost of maintaining the transformation tools, especially with the continuous change in software technology

6 difficulty in synchronising changes that might raise conflicts that require human interaction to resolve it

7 no possibility for concurrent collaboration on the product model.

In the current CAX integration approach, an adapter or a custom built component must be built that takes care of the translation when exchanging information between two systems, and the adapter must be compatible with a middleware and use the same language for communication. This work proposes two DSLs for PLM as the medium of communication. The languages' specifications combined with a hub architecture (adapters are required) to provide standardised automated infrastructure to enable machine-to-machine interactions and leverage the collaboration aspects between different vendors contributing to the same product model. Figure 1 shows a framework at an abstract level that makes connections with the underlying geometry and topology of a particular feature (Tang et al., 2004).

Figure 1 A feature-oriented database framework for web-based CAx applications



This paper proposes a solution that is based on the new technologies of associative features (Ma and Tong, 2003), Web Services specifications (WS-*), SOA, and Extensible Markup Language (XML). The solution is aimed to result in the high acceptability and adoption by the industry while reusing much of the currently existing systems. In the rest of this paper, the architectural details of the system from the engineering contents communication point of view are discussed in Section 2. Then, in Section 3 the modelling language requirements are introduced. Sections 4 and 5 propose the specifications of two major components of the proposed solution, feature and PMLs, respectively. Section 6 offers concluding remarks.

2 Architectural overview

This research focuses on feature-based collaboration throughout product lifecycles. The process of exchanging CAX files involves file translation and transformation, and is limited to geometric information, hence leading to misinterpreting design intent and loss of information. Moreover, equivalent constructs for items translated must exist in the destination CAX tool. Even exchanging CAX files between the same CAX tools does not guarantee the recognition of constructs by the destination application. This research leverages the concept of associative feature (Ma and Tong, 2003) where the concept expands the feature definitions of specific application-related shapes through a set of well-constrained geometric entities. By using object-oriented approach, a feature type can be modelled in a declarative manner which basically consists of the properties and behaviours. Feature properties define the geometric entities, and behaviours define the related constraints and logics in functioning methods throughout the lifecycle of any feature instance. With the built-in object polymorphism capability, a systematic modelling scheme for a generic and abstractive parent feature class, with levels of specification as per application domain requirements, can be developed.

One major concern in the research community is how the architecture can be used to support a product development environment where multiple CAD systems (e.g., CATIA, NX, Pro/E or STEP) are present. Clearly, a neutral product modelling scheme has to be developed. This neutral product model makes use of a comprehensive mapping scheme that could interface the neutral and equivalent geometric and non-geometric entities with different CAX systems. While the information shared across different entities is associated via a feature-oriented product repository system (Ma et al., 2007), but the feature validity and functional methods to create, edit, and keep the integrated constraints, are expected to be handled via an application specific product modelling engine, such as the solid modellers at the bottom of CAD application architectures. For the geometric modelling aspect, the STEP standard data structures can be adopted as a common reference model. For more advanced feature-oriented applications, the centralised fine grain repository system (Ma et al., 2007) can be further designed and implemented to support the interactions and the reference relations across different applications at a higher level under a feature-oriented collaboration scheme.

Such a generic feature definition scheme unifies many traditionally defined (ISO, 1999, 2000), application-oriented feature definitions, and supports XML representation and fine grain database repository. Under the associative feature concept, the associative constraints across multiple phases of applications of a product's life cycle, complicated engineering patterns and engineering intent can be implemented. An example associative feature, cooling channel pattern in plastic injection mould design, was given in Ma and Tong (2003) (see Figure 2). An initial sketch-based conceptual pattern is implemented and its downstream cooling 'hole' features are derived from the pattern; and then the related assembly interfacing features and associated standard components at the manufacturing and assembly stages are associatively generated and managed via a well defined feature class model. Figure 3 depicts another example of associated feature, 'referenced face regions' which defines the associated geometric mapping between a rubber part region, its related core and cavity mould surface regions, and the corresponding region created on the electrode for manufacturing the cavity.

Figure 2 An example associative feature, cooling channel pattern

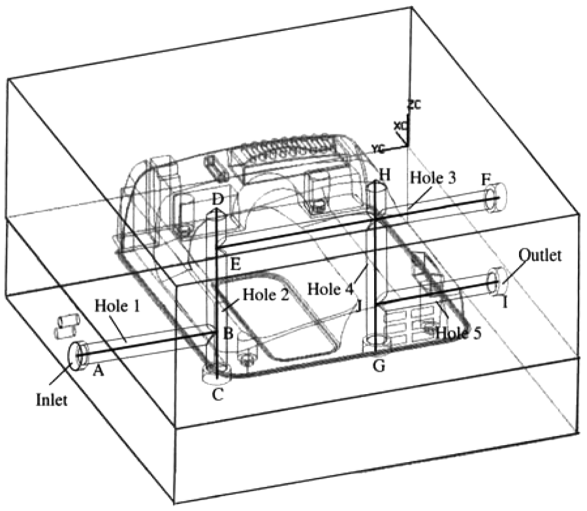
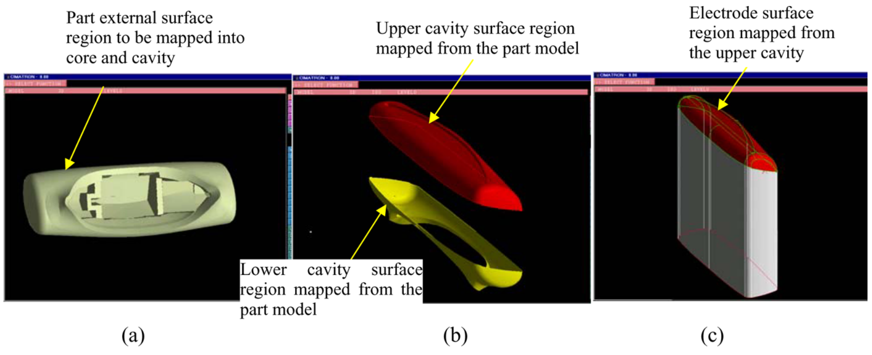


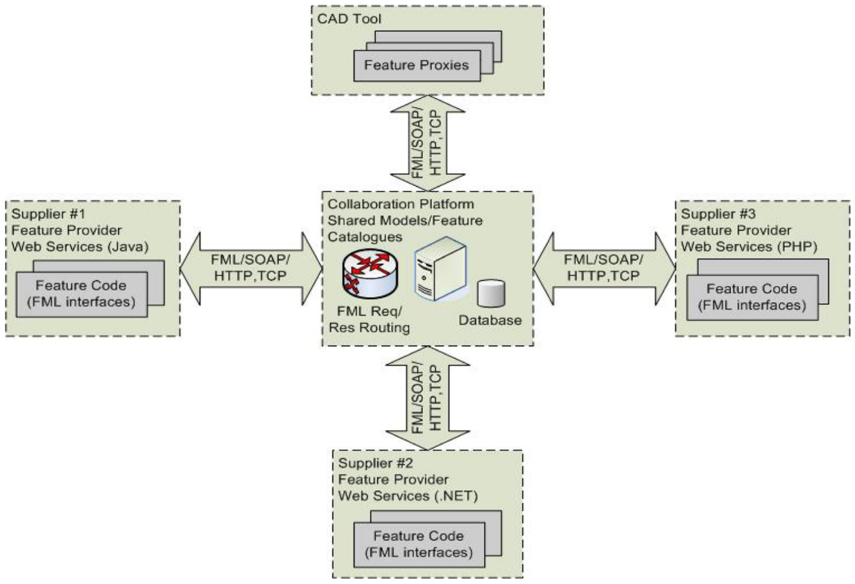
Figure 3 ‘Referenced surface region’ in an associated feature applied in a rubber mould design: (a) Rubber part model; (b) Extracted surfaces for the upper and lower cavities and (c) Electrode model developed (see online version for colours)


Part external surface region to be mapped into core and cavity
Upper cavity surface region mapped from the part model
Electrode surface region mapped from the upper cavity
Lower cavity surface region mapped from the part model
(a)
(b)
(c)

The associative feature concept does not emphasise the particular name of a traditional feature definition in the real application, such as a ‘pocket’, a ‘slot’, or a ‘hole’. Rather, it emphasises the overall support of the common properties and behaviours from the parent ‘generic feature’ for the common functions of any feature, and a well-defined sub-class of the ‘generic feature’ parent, i.e. the ‘pocket’, or the ‘slot’, or the ‘hole’ which supports the relevant properties and behaviours of their specific application properties and behaviours. In associative feature modelling, a feature name is only a labelling attribute that identifies a sub-class definition type. Feature information is utilised and shared by using OS and programming language independent constructs based on SOA. We approach the problem by defining a web-friendly and machine sensible modelling language to define features and exchange CAX information.

Figure 4 is a conceptual architecture diagram of the solution. The core application component is a collaboration server supported by middleware that exposes a set of web services. The server manages the flow of model information between collaborating partners. The collaboration server manages the feature definitions catalogue. Feature definitions are articulated using FML which is based on XML and part of our PLM-DSL. FML is further discussed in Section 4. Suppliers collaborating on the same model are themselves service providers by exposing a set of web services that invoke and execute their feature logic implementation codes.

Figure 4 Conceptual architecture diagram (see online version for colours)



A CAX tool modifies a model by invoking some operations or methods on certain features. The invocation action goes through a so-called feature proxy which passes any real invocation to the collaboration server. The collaboration server using a routing mechanism would connect to the feature provider and channel through the invocation. The logic is executed on the feature provider server and the invocation result is returned to the CAX tool through the collaboration server. This architectural design frees the collaborators from exchanging libraries that include the feature implementation. A consumer of a feature interacts with the actual implemented feature through feature proxies that act as stubs that channel the method invocation through the collaboration platform to the provider of the feature. On the other hand, any change on the model is propagated to the CAX data models and managed by the collaboration server. Subsequent changes are validated by the collaboration server to ensure the consistency of model through checking the constraints involved at the model and feature levels. Systems communicate with the collaboration server using PLM-DSL constructs on top of Simple Object Access Protocol (SOAP) messages. SOAP messages can run on top of different transport protocols such as HTTP, TCP, and UDP. WS-* specifications such as Reliable Messaging and Transaction can further be utilised to enhance the communication between the CAX systems and the collaboration server.

3 Modelling language

The product lifecycle spans multiple phases and involves different contributors with different roles during PLM (Thimm et al., 2006; Ma and Fuh, 2008). PLM requires the collaborative creation, management, dissemination, and the use of product model and process model information across the extended enterprise, from market concept to product retirement. This collaborative nature mandates the CAX information sharing between partners. Non-geometrical information and design intent are stripped away during the process of CAX file exchange via some format such as STEP standard (ISO, 1999, 2000). As a consequence, the major challenge is to develop a 'feature-centric' expert system that incorporates knowledge and information from all phases of the PLM and supports global, concurrent design and engineering. Ma et al. (2005), Ma and Tong, (2000) introduced the concept of Associate Feature (AF) as a form of self-contained and well-defined design object to capture the semantics and intents during the product lifecycle. However, their previous works described AF more in terms of its constituent elements and form.

This work proposes a neutral modelling language to model a product using fine-grained AFs as the building blocks. It leverages the collaborative environment by capturing the semantics and design intent during the product lifecycle. Moreover, by increasing the level of abstraction, the proposed architecture breaks away from a specific operating system and programming language. These criteria can be met by developing a DSL for the proposed modelling method (Regio et al., 2005). Therefore, the objective of this work is to design DSL language at different levels of information abstraction to capture the information flow in PLM. Some of the considerations undertaken in the design of PLM-DSL include

1 modelling both geometric and non-geometric information

2 web-friendly mechanisms to enable collaboration and data transfer over the web

3 machine readable and computer sensible data types and methods to enable automation and machine-to-machine communication

4 extensibility and composability to accommodate for the evolving nature of the industry

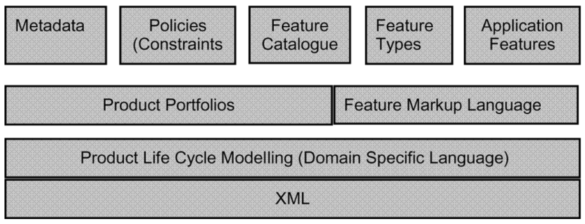
5 versioning support especially for the AF definitions as well as their instances such that AF life can be extended for many years.

The proposed DSL schema needs to be flexible enough to preserve the semantics of the artefacts and to allow some degree of extensibility to meet the needs of growing deployment.

XML is used as the underlying language for designing the Domain Specific language. Web services and web service specifications are to be applied to enable automation of scenarios related to integration and B2B transactions. XML is machine, OS and programming language independent and, moreover, many tools and libraries exist in the market nowadays to process XML documents. PLM-DSL is composed of two protocols to meet our requirements for modelling: FML and PML. Both of them are expressed as XML schemas using the XML schema definition language, 'XSD 1.1'. The schemas define the grammar, structures and types in an XML document (Thompson et al., 2004; Biron et al., 2004). Using XSD offers many advantages, e.g., it is machine readable;

those defined schemas are self documented; it is a W3C recommendation and widely accepted as a standard; and its definitions can be versioned. Figure 5 depicts the building blocks of the proposed PLM-DSL.

Figure 5 Product modelling language



4 FML

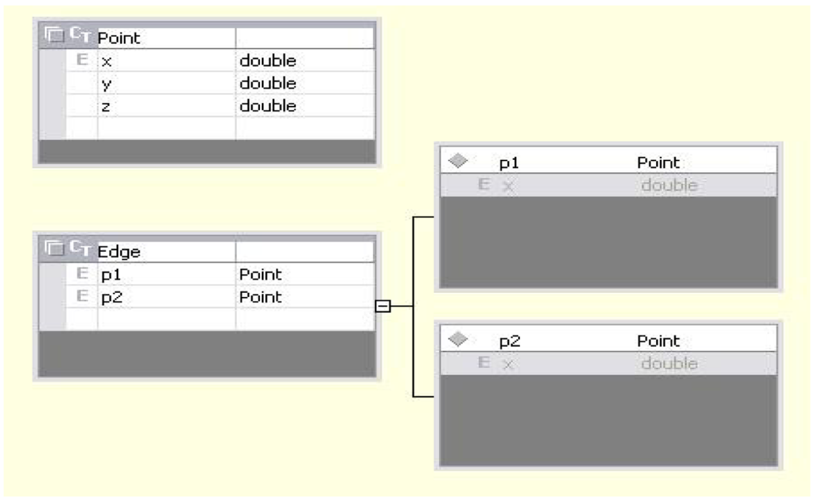
Associate Features are fine-grained objects that encapsulate both data and behaviour (Ma et al., 2007). The data encodes the semantics and state of the feature. Behaviours are realised in the form of methods and if invoked would change the internal state of a feature. FML was first introduced in Ma et al. (2009) as an associative feature modelling language. FML schemas are based on XML Schema Definition (XSD 1.1). They define the grammar and constructs of FML constituents. An FML document is an XML info-set conforming to the FML schema that captures both the geometric and non-geometric details of a feature preserving the semantics and intent of the designers. An FML schema describes the attributes, constraints and the interfaces of a feature. The following sections describe these concepts in further detail.

4.1 Attributes

Attributes represent the data state of the feature and are identified by unique names within the FML definition. An attribute type can be as simple as a primitive type such as integers, strings and float, or complex types such as points and faces structure. Complex types are aggregates of other simple and/or complex types. XSD schemas provide the capability to define a rich set of types that are reusable by other XSD schemas. Such reuse of schema defined types is analogous to reusable types packaged in software libraries such as dynamic link libraries in the windows operating system world.

Figure 6 graphically represents a complex type 'Point' of a schema for complex types. Type 'Point' having three primitive types *x*, *y* and *z* of type double. Another complex type recognised from the figure is the type 'Edge' which in turn has two attributes p1 and p2 of type 'Point'. To demonstrate the type definition capabilities of XSD, we defined XSD types that are equivalent to the topological geometric objects of ACIS. ACIS is a geometric modeller which represents a shape in terms of a network of interrelated geometric and topological objects (Corney and Lim, 2001). The reason for choosing ACIS as a reference data structure model is the capability of cellular representation that allows multiple views of geometrical entities to coexist without conflicts. On the other hand, any solid modeller that has a mechanism to represent unified (especially geometric) entity multi-views can be considered as a candidate reference model. Eventually, a mapping mechanism that can encapsulate different ‘engines’ into a ‘software factory’ has to be studied and developed.

Figure 6 Complex type definitions (see online version for colours)



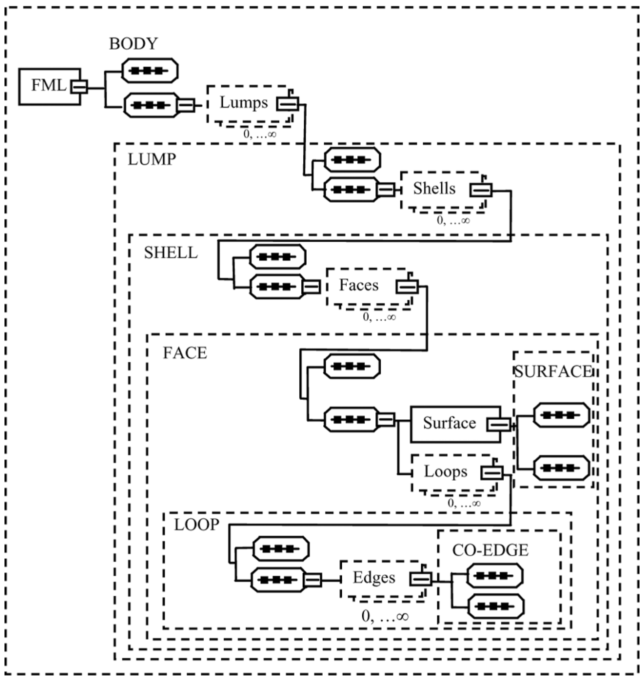
Listing 1 is the XML schema showing the defined types of both Point and Edge as complex types. XSD schemas include a couple of mechanisms both data-centric and object-oriented for creating and defining a rich type system. Some of the data-centric concepts include constraints such as required values and unique keys while concepts such as inheritance are inferred from the object-oriented approach.

Listing 1 Schema complex type definition (see online version for colours)

```
<?xml version="1.0" encoding="utf-8"?>
  <xs:schema        targetNamespace="http://tempuri.org/XMLSchema.xsd"
elementFormDefault="qualified"
xmlns="http://tempuri.org/XMLSchema.xsd"
xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="Point">
  <xs:sequence>
   <xs:element name="x" type="xs:double" />
  <xs:element name="y" type="xs:double" />
  <xs:element name="z" type="xs:double" />  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="Edge">
  <xs:sequence>
   <xs:element name="p1" type="Point" />
   <xs:element name="p2" type="Point" />  </xs:sequence>
 </xs:complexType> </xs:schema>
```

Figure 7 depicts one way to model ACIS objects using XSD type system. Many of the type attributes are omitted, to simplify the graph. Every defined type inherits directly from the ENTITY type. The LUMP type, for example, is defined as a complex type that has a collection of SHELL objects.

Figure 7 ACIS representational hierarchy using XML schema definition



4.2 Constraints

Constraints encode the rules and policies of a feature and define the relationship with other features in a product model. Some examples of constraints are parameter constraints such as dimensions, relations of referenced entities, tolerances, while some are more complicated and to be managed by certain procedures, such as derivations from upstream entities, and evolved design patterns like cooling circuit in mould design (Ma and Tong, 2003). The constraint knowledge is encapsulated in a feature and controls the modifications applicable to its internal state. There are several levels where constraints can be defined

1 At the type level; the acceptable set of values the type can assume is defined. This is implemented through XSD type system.

2 At the feature level which governs the rules and policies that relates attributes within the feature. The rules are defined at the FML level using the FML schema constraint constructs.

3 At the model level which governs the relationship between different features. The constraints are defined at the model level using PML schema constructs.

4.3 Interfaces

As mentioned earlier, a feature encapsulates behaviour that defines the set of operations callable on the feature. An operation is a method that can be invoked on a feature and leads to one of the following effects:

1 changing the internal state of the feature by modifying one or more attribute value

2 changing the state of other features in the model governed by the constraint set

3 creating and deleting an instance of a defined type.

To allow user defined features, FML schema enables feature developers to define behaviour by using interfaces. An interface is a group of related operations that specify an abstract contract. The interface lists the methods defined in a feature and the input parameters passed to a feature and the returned output parameters. It does not provide a description of the concrete implementation of the feature. An interface is similar to the port construct in a Web Service Description Language (WSDL) document. WSDL is an XML based language and was the first widely adopted mechanism for describing the basic characteristics of a Web service (Cabrera and Kurt, 2005). A WSDL document describes how to call a method in a service without describing the internals of the method implementation. It describes what a request message must contain and what the response message look like in unambiguous notation. The Grouping of operations into interfaces allows feature developers to reuse the interface definitions during the development of other features and in such a way a type can be reused.

Listing 2 is an FML fragment that defines an interface called IFace and has two operations: CountEdges and AddEdge. The CountEdges operation has no input parameters, while it returns an output parameter of type integer. On the other hand, the AddEdge operation accepts one input parameter of type EDGE and returns an output parameter of type Face.

Listing 2 FML interface definition (see online version for colours)

```
<interface name="IFace">
  <operation name="CountEdges">
    <out type="integer"></out>
  </operation>
  <operation name="AddEdge">
    <in name="edge" type="EDGE"></in>
    <out name="face" type="FACE"></out>
  </operation>
</interface>
```

A feature implementing the interface IFace inherently acquires the behaviour of a surface and supports the operations CountEdges and AddEdge. Note that the IFace interface is defined as a XSD type and therefore can be applied to other features. To successfully call an operation defined by an interface, the application must make sure to pass the input parameters as defined by the interface. A feature can have 0 or more interfaces depending on their complexity. The more interfaces implemented by a feature the more it has support for operations. We can define unlimited number of interfaces using the same type definition mechanism of XSD and as dictated by the FML language. Consider a scenario where we need to add extra information to a feature. In this case we can define an interface called ITaggable, that has one method called GetTags, and apply the interface to the feature. The GetTags will return the extra information associated with the feature. The mechanism of applying interfaces to features allows feature developers to tag their features with interfaces using interface definitions. Listings 3 and 4 demonstrate the concrete implementation of an interface using C# and C++, respectively.

Listing 3 C# sample code implementing IFace

```
interface IFace{    int CountEdges();

         void AddEdge(EDGE edge); }

class SampleFeauture : IFace { Array _edges;
         int CountEdges()    {

          return _edges.get_count();    }
         void AddEdge(EDGE edge) {
                  _edges.add(edge); ;};
```

Listing 4 C++ sample code implementing IFace

```
class IFace {
            virtual int GetCount();
            virtual void AddEDGE(EDGE* edge); };

class SampleFeature : IFace{
            private:  int _count;
            public:   int GetCount() {
                                /*logic*/ };
            void      AddEDGE(EDGE* edge) {
                                /*logic*/ };  };
```

The code implementation of the IFace interface corresponds to the schema defined previously. A class named SampleFeature implements the interface by providing the full method implementation.

A CAX tool using the SampleFeature needs to know only the lists of interfaces defined by the feature. If the CAX tool identifies that a feature implements the IFace interface, that tool can invoke the methods CountEdges and AddEdges. This invocation of methods, combined with the collaboration server architecture discussed earlier, is actually channelled all the way to the feature provider using SOAP based messages. FML is the underlying protocol that ensures the proper discovery of the interfaces, the methods exposed by an interface and the messages required to invoke these methods.

An interface outlines the generic structure of associative feature; it avoids the concrete implementation of the behaviour to achieve a greater level of abstraction. This abstraction, enables suppliers and feature developers to extend the feature schema to create new features to cover their needs. However, the FML schema draws the general guidelines and rules for developing features to ensure the consistency and modularity of the developed features.

A FML instance is an XML document based on an FML schema. An analogy can be drawn between an XML document and an FML schema and an object to a class in object-oriented programming. The XML document encapsulates the parameterisation of the feature. An XML document always refers to the schema and is validated against it to ensure the validity of the XML document structure. Schema referencing is a way to uniquely identify schemas published by different suppliers through the use of Uniform

Resource Identifiers (URI) (refer to http://www.supplier.com/schemas/v1.0/fml.xsd). The reference scheme creates some kind of namespace, i.e., a set of uniquely defined coded names, for the features defined in the schema. The namespace helps identify features developed by different suppliers that have the same schema definition.

The FML manages the collection of features and their internal data structures. The collaboration server constantly ensures the consistency of the internal data state of features through the evaluation of the constraints. Different views can be created to synchronise certain information from the feature collection. For example, geometrical information can be extracted in a machining feature view by certain feature recognition algorithms (Han et al., 2000) or interactive identification by the machinist to create a machining sub-model while stripping away irrelevant information from design stage.

5 PML

This proposed Product Modelling Language encodes the modelling information during the product lifecycle. Its format is also based on XML which makes it readable by machines and suitable for data exchange using web services. PML defines an XSD schema that lays out the structure of the XML file encoding the product lifecycle details. The XML file holds both geometric and non-geometric information. The definitions of typical geometric information represented in the form of features, solids, faces, edges, vertices, as well as their geometrical shapes or positions, are well reported in the literature. The representation of non-geometric information is relatively less studied; a related review is available (Ma et al., 2008a, 2008b).

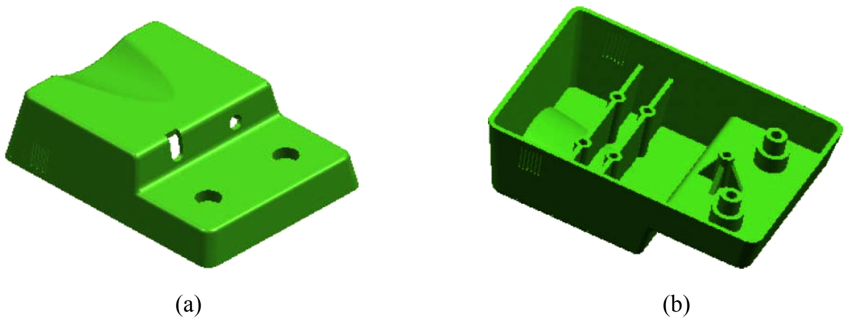
Referring to Figure 5, the Product Modelling Language consists of the following three building blocks.

1. Metadata is defined as data describing data. Therefore, PML Metadata provides the means to further associate data to the product model. Create date, authors, CAX tools and CAX versions are just a few examples of metadata that are associated with the model. The metadata system in PML is extensible, allowing the collaborators to contribute to the semantic of the model.
2. Model constraints (policies) are part of the constraint space defined in a model. PML constraints govern the relationships between the set of features in a model. These constraints are constantly evaluated to ensure the consistency of the model data state.
3. A Feature Catalogue lists the available feature definitions in the product model. This list includes the subset of features' definitions used in the product model and in addition to other features definitions available to model designers. The CAX tools explore this catalogue to identify the used features and to generate features' proxies. The features' specific proxies are eventually used by the CAX tools to contribute to the product model created and managed at the collaboration server level.

5.1 Software factories

FML is useful for building software factories to automate the generation of proxy features components and libraries. According to (Greenfield et al., 2004), a Microsoft software factory can be defined as a production line that configures extensible development tools like Visual Studio (Guckenheimer and Perez, 2006), with packaged contents and guidance, designed for building software applications. An initial investment in building software factories targeting different CAX tools can cut cost and time in building the proxy features. A CAX vendor can provide the tools on top of his product to 'consume' FML documents and generate the features corresponding to the FML document. This process can be automated (Chen et al., 2005) and reduces the time and effort to build proxy features. Figure 8 shows the test case that has been created and modified using the common 'operations' but executed on both Siemens NX (formerly known as Unigraphics) and SolidWorks CAD systems.

Figure 8 An example part that has been created and modified on both Unigraphics and SolidWorks systems by using 'operations' (see online version for colours)



(a) (b)

Source: Chen et al. (2005)

6 Conclusion

Based on the default mechanisms of XML over the internet that support the multimedia interoperability and granularity, this research work promotes the idea of FML and PML as the domain specific languages for product lifecycle modelling. These languages enable neutral, content-driven and computer-interpretable communication solution that can effectively support 'encapsulation' of low level entities that are currently being generated and used by the existing CAX systems. A service-oriented architecture was proposed to leverage the capabilities of these languages and create an agile environment for collaboration and information sharing. This work describes a new communication approach for concurrent collaboration at a finer grains-associative feature level than previous works. For future research directions, the authors believe in the opportunities of constraint management (Ma et al., 2008a, 2008b) and engineering intent modelling.

References

Biron, P.V., Permanente, K. and Malhotra, A. (2004) *XML Schema Part 2: Datatypes*, 2nd ed. (Online) Available: http://www.w3.org/TR/xmlschema-1/

Cabrera, L. and Kurt, C. (2005) *Web Services, Architecture and Its Specifications: Essentials for Understanding WS-**, 1st ed., Microsoft Press, Washington.

Chen, J.Y., Ma, Y-S., Wang, C.L. and Au, C.K. (2005) 'Collaborative design environment with multiple CAD systems', *Computer-Aided Design and Applications*, Vol. 2, pp.367–376.

Corney, J. and Lim, T. (2001) *3D Modeling with ACIS*, 2nd ed., Saxe-Coburg Publications, Stirling, UK.

Greenfield, J., Short, K., Cook, S., Kent, S. and Crupi, J. (2004) *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley and Sons, Toronto.

Guckenheimer, S. and Perez, J.J. (2006) *Software Engineering with Microsoft Visual Studio Team System*, 1st ed., Addison-Wesley Professional, Upper Saddle River, NJ.

Han, J.H., Pratt, M. and Regli, W.C. (2000) 'Feature recognition from solid models: a status report', *IEEE Transaction on Robotics and Automation*, Vol. 16, pp.782–796.

ISO (1999) *STEP – Part 224: Application Protocol: Mechanical Product Definition for Process Planning Using Machining Features*, ISO 10303-224, International Organization for Standardization (ISO), Geneva, Switzerland.

ISO (2000) *STEP – Part 42: Integrated Generic Resource: Geometric and Topological Representation*, ISO 10303-42, International Organization for Standardization (ISO), Geneva, Switzerland.

Ma, Y-S. and Fuh, J.Y.H. (2008) 'Product lifecycle modelling, analysis and management', *Computers in Industry*, Vol. 59, pp.107–109.

Ma, Y-S. and Tong, T. (2003) 'Associative feature modeling for concurrent engineering integration', *Computers in Industry*, Vol. 51, pp.51–71.

Ma, Y-S., Britton, G.A., Tor, S.B. and Jin, L.Y. (2005) 'Associative assembly design features: concept, implementation and application', *International Journal of Advanced Manufacturing Technology*, Vol. 32, pp.434–444.

Ma, Y-S., Chen, G. and Thimm, G. (2008a) 'Change propagation algorithm in a unified feature modeling scheme', *Computers in Industry*, Vol. 59, pp.110–118.

Ma, Y-S., Chen, G. and Thimm, G. (2008b) 'Paradigm shift: unified and associative feature-based concurrent and collaborative engineering', *Journal of Intelligent Manufacturing*, Vol. 19, pp.625–641.

Ma, Y-S., Jiao, J. and Deng, Y. (2009) 'Web service oriented electronic catalogs for online product customization', *Concurrent Engineering: Research and Application*, Vol. 14, pp.263–270.

Ma, Y-S., Tang, S-H. and Chen, G. (2007) 'A fine-grain and feature-oriented product database for collaborative engineering', in Li, W.D., Ong, S.K., Nee, A.Y.C. and McMahon, C. (Eds.): *Collaborative Product Design and Manufacturing Methodologies and Applications*, Springer-Verlag, London, pp.109–134.

Regio, M., Greenfield, J. and Thuman, B. (2005) *A Software Factory Approach to HL7 Version 3 Solutions*, (Online) Available: http://msdn2.microsoft.com/en-us/library/ms954602.aspx

Tang, S-H., Ma, Y-S. and Chen, G. (2004) 'A feature-oriented database framework for web-based CAx applications', *Computer-Aided Design and Applications*, Vol. 1, Nos. 1–4, pp.117–125.

Thimm, G., Lee, S.G. and Ma, Y-S. (2006) 'Towards unified modelling of product life-cycles', *Computers in Industry*, Vol. 57, pp.331–341.

Thompson, H., Beech, D., Maloney, M. and Mendelsohn, N. (2004) *XML Schema Part 1: Structures Second Edition* (Online) Available: http://www.w3.org/TR/xmlschema-1/