INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'09
24 - 27 AUGUST 2009, STANFORD UNIVERSITY, STANFORD, CA, USA

VALUE – FUNCTION – STRUCTURE MODELING IN AN ONTOLOGICAL REPRESENTATION OF PRODUCT-SERVICE SYSTEMS

Yong Se Kim[1], Eric Wang[1], and Myon Woong Park[2]
[1] Creative Design Institute, Sungkyunkwan University, Korea
[2] Korea Institute of Science and Technology, Korea

ABSTRACT
An ontological representation of product-service systems (PSS) is presented, which could support the development of new computer-aided tools and frameworks for PSS design. In this ontology, a PSS consists of values, product and service elements, and their relations. PSS and its product and service elements are further specified by attributes of function, behavior, structure, context, and environment. This paper further describes an ontological representation of the mappings between values, functions, and structures. This is developed as an ontology of generic decomposition knowledge, which can express an abstraction/realization hierarchy across multiple levels of abstraction, in which an element at a higher level (more abstract) is mapped to one or more sets of elements at a lower level (more detailed). The same ontology is also applied to represent multiple levels of abstraction in a function decomposition of products within a PSS.

Keywords: Abstraction/Realization, Function Decomposition, Ontology Modeling, PSS Representation

1 INTRODUCTION
Product-service systems (PSS) provide a means of innovative value proposition through the integration of products and services [1]. PSS is part of an ongoing trend away from a solely product design-oriented focus, to embrace the entire lifecycle of products, and to explicitly design and manage new service applications such as maintenance and rental that are enabled by having these products, to create new business opportunities and/or reduce environmental costs. In this way, manufacturers are evolving into service providers. Baines *et al.* review research in developing the tools and methodologies for effective development of PSS [2]. Goedkoop *et al.* define PSS as "a marketable set of products and services, jointly capable of fulfilling a client's need" [3], while Mont defines PSS as "a system of products, services, supporting networks, and infrastructure that is designed to be competitive, to satisfy customer needs, and to have a lower environmental impact than traditional business models" [4][5].

While PSS is currently an active research topic, smart tools to support PSS design (e.g. comparable to existing CAD systems for product design) have yet to mature. An overall goal of this research is to develop a formal, ontological representation of a PSS, and of the PSS design process itself, which would be a foundation for a PSS design tool. Ontology is an emerging semantic technology for computer-understandable representations of the entities, relations, and constraints in a domain [6]. We use standard ontology tools, including OWL ontology language [6] and Protégé ontology editor [7].

In designing a product or service, it is crucial to anticipate the needs of customers and other stakeholders. In PSS research, stakeholders' needs are identified, and are represented as *values*. While value in the economic sense is "the market worth or estimated worth of products or services" [8], each individual value in PSS design is generally a concise term or phrase that defines a specific, measurable aspect of a product or service that has "economic value" to a stakeholder. Shimomura *et al.* introduce the service model in service engineering, and propose receiver state parameters (RSP) as a quantitative implementation of values [9][10][11][12][13]. They use Kahle's List of Values [14] as the highest abstraction of receiver values, and decompose these values hierarchically to obtain RSPs.

Maussang *et al.* present a PSS design process that incorporates Shimomura's service model with an engineering product design process, considering functional analysis and agent-based value design, in which values are deduced from the needs of stakeholders [15]. Using this, they develop a PSS for the Vélo'v self-service bicycle rental system, incorporating bicycles, stations with locking facilities, rental services, and maintenance operations.

The rest of this paper is organized as follows. Section 2 summarizes our ontological representation of PSS. Section 3 discusses an abstraction/realization hierarchy between the three levels of *value*, *function*, and *structure*, and reviews existing research in function decomposition and related approaches in product design. It presents a generic ontology of mapping between two sets of elements, and discusses how it is merged with the PSS ontology. Section 4 elaborates on the representation of *function* in PSS. Section 5 shows an example of a function decomposition, and its implementation using our generic mapping ontology.

2 ONTOLOGICAL REPRESENTATION OF PSS

In parallel with the current research, we are also developing an approach for the conceptual design of PSS, based on established product design methodologies and a wide literature review of PSS design research. A case scenario of this approach to develop a PSS for a meal assembly kitchen has been presented in [16]. From that scenario, we have identified the elements of a PSS representation, using a simple graph representation. We formalize this representation as an ontology of a PSS, which describes values, product elements, and service elements, and their relations, as shown in Figure 1. We use OWL and Protégé for ontology modeling, with UML for presentation.

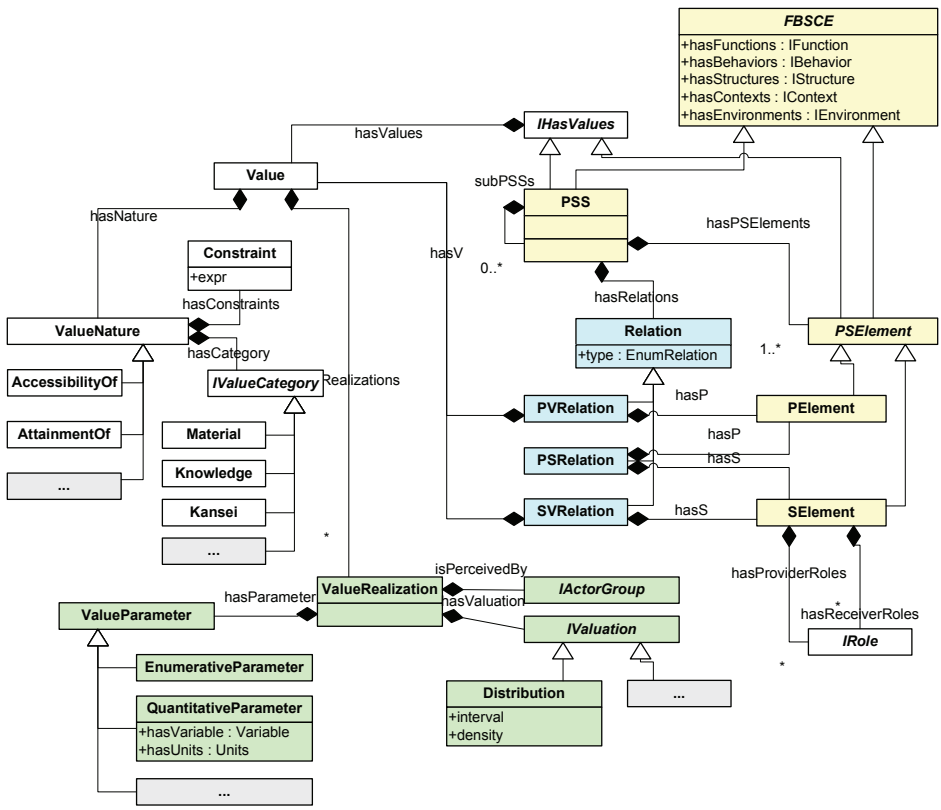

Figure 1. Ontological Representation of Value and PSS

2.1 Representation of Value

While *values* have been proposed in numerous design methodologies for service [9]–[13] and PSS [15], existing descriptions of values are ad hoc, using natural language phrases. The current work pursues a formal ontological representation of value. By generalizing from numerous examples of values in the cited literature, and those we developed for our case scenario in [16], we have developed an initial representation of a **Value** class, shown on the left side of Figure 1. Its constituents are:

- one **ValueNature** object that describes *what the value is*;
- multiple **ValueRealization**s that represent different *subjective interpretations* of this value.

Each **ValueRealization** consists of:

- an *IActorGroup* for a subset of stakeholders that are characterized by some shared attributes;
- a **ValueParameter** that records a measurement of a property of this actor group;
- an ***IValuation*** that maps the value parameter to these actors' "degree of satisfaction".

The **Distribution** subclass is one kind of valuation, which requires that its state parameter be quantitative, and defines a mathematical function from parameter values to "degree of satisfaction".

2.2 Representation of PSS

The ontological model of PSS is shown in Figure 1, upper right. A **PSS** is represented as a class which aggregates three constituent classes, representing a set of **Value**s, a set of ***PSElement***s, and a set of **Relation**s of this PSS. Furthermore, **PSS** uses a notion of an abstract base class that includes *function*, *behavior*, *structure*, *context*, and *environment* attributes, which is named ***FBSCE***. PSS also has a **subPSSs** relation to 0 or more other PSSs, which allows a PSS to be composed from sub-PSSs in a recursive manner.

The ***PSElement*** class is an abstract base class, whose subclasses represent the elements of a PSS. It conveniently extracts common operations from its subclasses, which simplifies numerous algorithms. ***PSElement*** also inherits function, behavior, structure, context, and environment attributes from the ***FBSCE*** base class.

- **PElement (product element)** describes a product design.
- **SElement (service element)** describes a service, including its provider and receiver roles.

Values are associated with product elements, service elements, and PSSs. To ensure a consistent interface, we define an abstract interface *IHasValues*, which carries a hasValues property to 0 or more Value objects. We then add *IHasValues* as an additional base class of both **PSS** and **PSElement** classes, using multiple inheritance.

Relation represents a relation between two elements, or between a value and an element. We define three specific subclasses, in order to exploit exact type information in modeling these subclasses, respectively:

- **PVRelation** between a product element and a value;
- **SVRelation** between a service element and a value;
- **PSRelation** between a product element and a service element.

3 REPRESENTATION OF VALUE – FUNCTION – STRUCTURE

3.1 Function Decomposition

Function decomposition is an established technique in product design and service engineering research; hence it could be incorporated as a cornerstone of PSS representation. In function-based design, the overall functions of a product are recursively decomposed into sub-functions, where all functions have input and output in the forms of energy, material, or signal [17]. Standardized sets of function and flow (energy, material, signal) classes have been established for products [18][19].

Function decomposition has been incorporated into more comprehensive design frameworks, including Function-Behavior-Structure (FBS) [20], Function-Behavior-State [21], and Function and

Behavior Representation Language (FBRL) [22]. These frameworks introduce a multi-level representation, where each level represents a product (or service) at a certain level of abstraction, and mappings between elements of different levels denote relationships such as abstraction/refinement, *realization-of*, *ways-of-achieving* [22], etc. In these frameworks, *function* represents the designer's intent, *behavior* is a realization of function using purely objective descriptions, and *structure* is a realization of behavior using concepts, physical components, and other product elements. Camelo *et al.* proposes a similar representation with 4 levels, with *purpose function* for the designer's intent, *action function* as an abstraction of behavior, *behavior* as an abstraction of physical state, and *structure* as an abstraction of geometry [23], where their *is-abstraction-of* (downward) relationship could be viewed as the inverse of the *is-realization-of* (upward) relationship. Such multi-level representations have been found to be useful in organizing design knowledge [24].

3.2 Representation of Multiple Levels of Abstraction

This paper focuses on representing the interrelations between *value*, *function*, and *structure* elements, organized into three layers, as shown in Figure 2. To extend the FBS-like design frameworks to accommodate PSS, *value* is added as the topmost level of abstraction. As values are deduced from the needs of stakeholders [15], it could be viewed that "*value* is the stakeholder's goal", and "*function* is a realization of the value" (by the designer, for the stakeholder). For this discussion, *behavior* has been abstracted away, and structure is considered to be "a realization of the function".

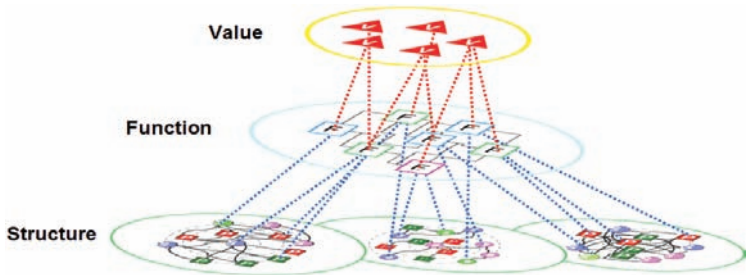

Figure 2. Mappings between Value, Function, and Structure Levels

An example of this multi-level abstraction hierarchy is given in Figure 3. In this example, value v1 is realized by a set of two functions { f2, f3 }. Function f3 can be realized in two alternative ways: by service { s9 } alone, *or* by the set of { product p5, service s6 }.

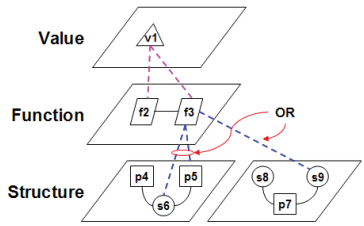

Figure 3. Example of Value-Function-Structure Hierarchy

Hence, the interrelations between elements in two levels of abstraction can include both one-to-one and one-to-many relations (and, by generalization, many-to-many relations). Also, a mechanism is needed to organize sets of interrelations, and express set-level operations such as alternatives (OR).

From these criteria, we have implemented a generic ontology of a **Mapping**, shown in Figure 4, which can model a set of connections between two sets of elements. To support diverse interpretations of the connections in a mapping, the *role* of each connection is defined separately, as an enumeration class, shown in Figure 4, lower right, in yellow.

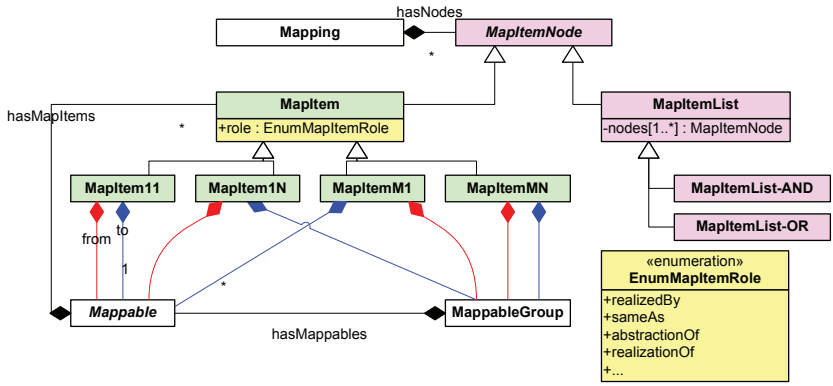

Figure 4. Generic Representation of Mappings as Connections between Elements

- The ***Mappable*** abstract base class (lower left, in white) denotes the capability of participating in a mapping, i.e. of being an endpoint of a connection. To merge this ontology with the PSS ontology, ***Mappable*** is added as an additional base class of the PSS ontology classes **Value** and **PSElement** (in Figure 1), and **Flow** and **Function** (which are further described in Section 4).
- The **MappableGroup** helper class (lower right, in white) represents a set of 2+ mappable elements. *Example:* The subsets {f2, f3} and {p5, s6} are implemented as instances of this class.

- The **MapItem** class (center, green) represents one generalized edge, i.e. one connection between 2+ elements. It defines 4 subclasses to explicitly represent the cardinality of elements at each end.
    - **MapItem11** is a 1-to-1 connection (simple edge). *Example:* The f3–s9 relation is implemented as an instance of this subclass.
    - **MapItem1N** is a 1-to-many connection (hyperedge). This describes the case where one higher-level element is mapped to a set of *N* lower-level elements. *Example:* The v1–{f2, f3} and f3–{p5, s6} relations are instances of this subclass.
    - **MapItemM1** is a many-to-1 connection.
    - **MapItemMN** is a many-to-many connection.

To express sets of relations, and set-level operations such as alternatives, an AND/OR tree structure is incorporated into this ontology (upper right, in magenta).

- The ***MapItemNode*** abstract base class represents an AND/OR subtree of generalized edges.
- The **MapItem** class, which represents one generalized edge as described above, is also defined as a leaf (bottommost) node in the AND/OR tree structure. Hence, the simplest AND/OR tree would consist of just 1 **MapItem** edge, with no siblings and no children.
- The **MapItemList** subclass represents a set of 1+ ***MapItemNode*** child nodes, in a recursive manner. This produces a tree structure whose leaf nodes are all **MapItem**s. Two subclasses are defined to represent the semantics of set-level operations:
    - **MapItemList-AND** means the union of its child nodes.
    - **MapItemList-OR** means that, of its list of child nodes, exactly one shall be selected, and all others are excluded. *Example:* The OR-operation between the f3–{p5, s6} and f3–s9 relations is implemented as an instance of this subclass.

Finally, the **Mapping** class (top left, in white) consists of a set of ***MapItemNode***s. This encompasses everything from 1 simple edge to a rich set of AND/OR trees of hyperedges. *Example:* The mapping shown in Figure 3 is implemented as a **Mapping** instance with two child nodes: (1) a **MapItem1N** instance for the v1–{f2, f3} relations shown in magenta, and (2) a **MapItemList-OR** instance for the blue relations involving f3, as described above. Note that the top-down organization in the abstraction hierarchy is flattened into sibling nodes in the AND/OR tree structure.

3.3 Implementation Issues

Every **MapItem** is defined to point to 2 or more ***Mappable*** elements. To simplify data structure traversal algorithms, it is desirable to maintain a doubly-linked data structure. Hence, every ***Mappable*** element shall also point back to all **MapItem**s in which it participates. This requires computational support to ensure that all ***Mappable*** and **MapItem** pairs remain properly doubly-linked. Such an approach is already widely used, e.g. in maintaining a boundary representation in a solid model, so it is expected to be straightforward.

While computational efficiency issues are not a primary concern so far, efficiency could be improved by defining many parallel sets of subclasses of the **MapItem** class hierarchy, each of which makes a commitment to a specific level of abstraction, or to a specific subtype of ***Mappable*.** For example, a hypothetical **VFItem** subclass of **MapItem** could restrict its *from* and *to* property ranges to be **Value** and **Function**, respectively. This makes the ontology more precise, at the cost of a proliferation of subclasses. It presumes that the modeling formalism permits a subclass to restrict the range of an inherited property. OWL restrictions can express this, but most object-oriented languages (including Java and Jess) lack a similar mechanism.

4 FLOW, FUNCTION, AND FUNCTION DECOMPOSITION

Function decomposition also uses multiple levels of abstraction internally. In function-based design, a function is a transformation from input flows to output flows, where a flow is an energy, material, or signal [19][25]. By applying function decomposition, function chains are created for each input flow, and these function chains are aggregated, obtaining a function decomposition at a lower (more detailed) level of abstraction.

This approach is illustrated using the function decomposition of a take-out (disposable) coffee cup, which is being developed in a case study of a PSS for recycling coffee cups. This example focuses on one function of the disposable cup.

- At Level 1, shown in Figure 5 (top), the overall function is summarized as "store coffee for human drinking", and its input and output flows are identified.
- At Level 2, shown in Figure 5 (bottom), the overall function has been decomposed into five sub-functions, and the input and output flows have been distributed accordingly.

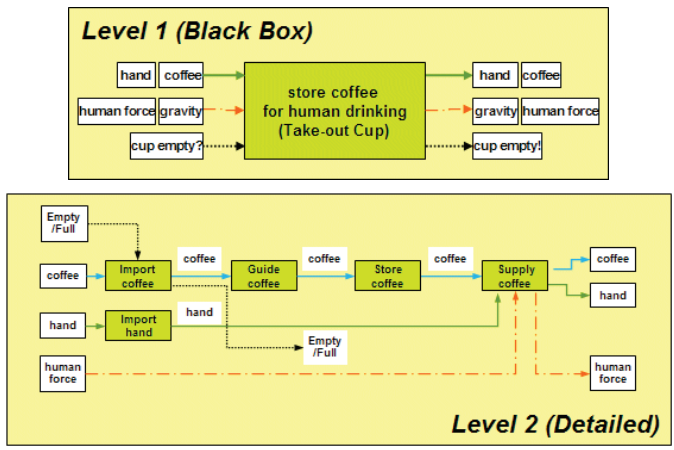

Figure 5. Function Decomposition with 2 Levels of Abstraction

Each flow is described in a compact and qualitative manner, i.e. all issues of quantization are abstracted away. From the signal flow instances of "empty" and "full", a notion of *state changes of flows* could be inferred as a result of the operation of the functions. However, state changes have not been fully incorporated into function-based design, and it could also be interpreted that "empty" and "full" are two distinct signals, not two states of one container product.

4.1 Representation of Flow

By drawing from established definitions in function-based design, and surveying recent literature in service functions used in service engineering, we have devised an ontological representation of flow and function, shown in Figure 6. A key concern in building this ontology was how to adequately model the instances of flows that appear as labels throughout each function decomposition. Each flow label represents a specific unit or quantity of a resource, with state, magnitude (if applicable), and other attributes; but at the high level of abstraction of function decomposition, most of these details are deliberately omitted, and only the *type* of the flow is stated. However, the ontological representation cannot be restricted to a high level of abstraction only. Hence, a mechanism is needed to allow a more abstract representation now, yet also support progressively more detailed representations in the future.

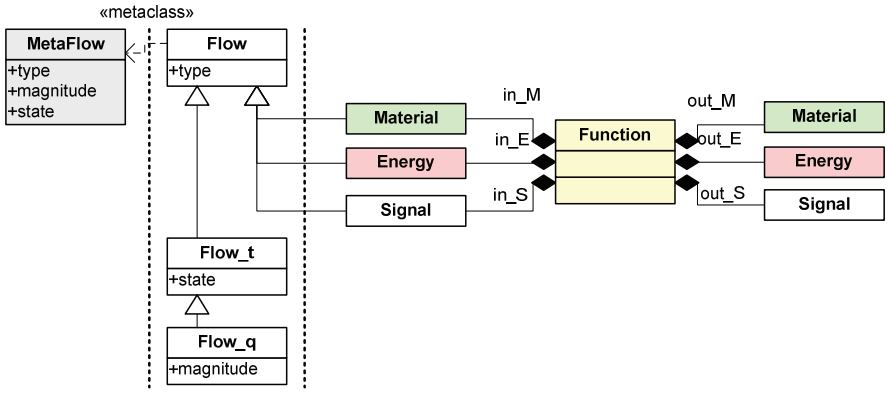

Figure 6. Ontological Representation of Flow and Function

The adopted solution is to use OWL metamodeling to support alternative Flow class hierarchies. The **MetaFlow** metaclass (upper left) specifies that attributes of *type*, *magnitude*, and *state* shall exist, but without committing to any specific implementation. Each instance of **MetaFlow** is a Flow class and its class hierarchy, which may implement these attributes differently among its subclasses.

One concrete Flow class hierarchy is then instantiated as shown:

- The **Flow** base class implements only the *type* attribute.
    - Its three subclasses of **Material**, **Energy**, and **Signal** each restrict the *type* attribute.
- The **Flow_t** subclass also implements the *state* attribute. This permits a richer description in which state changes of flows could be expressed.
- The **Flow_q** subclass also implements the *magnitude* attribute.

Function decomposition activities could initially use the **Material**, **Energy**, and **Signal** subclasses of **Flow** only. When detailed (quantitative) modeling is needed at a later stage, an ontology-aware PSS design tool could automatically upgrade instances of these classes to instances of the **Flow_t** or **Flow_q** subclasses to gain access to their additional attributes.

4.2 Representation of Function and Instantiation of a Function Decomposition

The **Function** base class is defined with six properties *in_{E,M,S}* and *out_{E,M,S}*, which represent any number of input and output **Energy**, **Material**, and **Signal** flows, respectively (including 0). This representation is illustrated by instantiating the Level 2 function decomposition for the "store coffee for drinking" example in Figure 5. The resulting ontology model is shown in Figure 7.

This example reveals an issue during the instantiation of the flow instances. In function decomposition, flow labels are not unique, and the same flow label may occur many times, without implying equivalence. In OWL, all individuals of the same class must have unique identifiers – and this applies to the flow individuals. The Protégé ontology editor uses a simple policy of automatically appending a numeric suffix that increases monotonically, which ensures uniqueness.

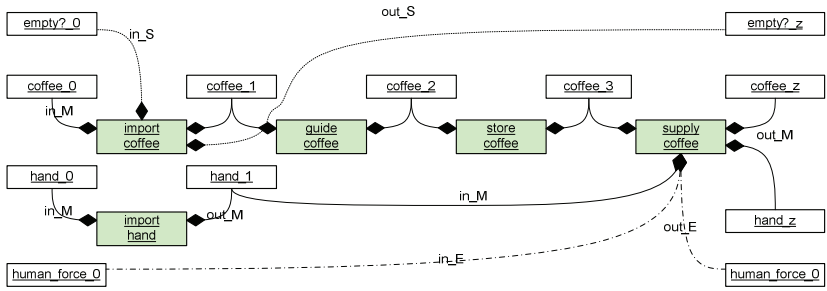

Figure 7. Instantiated Ontology Model of the Level 2 Function Decomposition

The following approaches to address this issue were considered.

- **Random suffixes.** Simple numeric suffixes could be misinterpreted as a kind of *state* information. This could be alleviated by adopting a policy of generating random suffix strings.
- **Shared flow individuals.** The need for numeric suffixes could be eliminated by instantiating only one **Flow** individual for each unique flow label, and sharing it among all **Function**s that include that flow label as an input or output. To preserve all knowledge of specific output-to-input pairs, a **Pair** helper class could be added that explicitly represents one output-to-input pair, and also references the **Flow** individual, as shown in Figure 8. However, this approach assumes that flow labels are entirely stateless, and that no loss of design intent occurs by merging identical labels.
- **Explicit state.** If, on the other hand, flow labels are intended to denote distinct states, and their apparent sameness is only an artifact of the high level of abstraction employed during function decomposition, then it may be desirable to instantiate them distinctly in OWL, and assign unique names. We could furthermore use a richer subclass such as **Flow_t** that provides additional attributes such as *state*, and rely on the design tool to automatically suppress them when the designer wishes to revert to a higher level of abstraction.

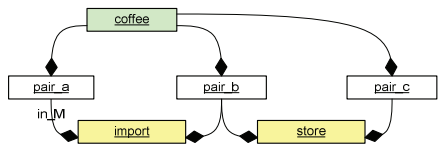

Figure 8. Shared Flow Individuals with Pair Helper Class

5 APPLICATION TO MULTIPLE LEVELS IN A FUNCTION DECOMPOSITION

In this section, the **Mapping** ontology of Section 3.2 is combined with the representation of **Flow** and **Function** in Section 4, and used to instantiate the 2-level function decomposition of the "store coffee for human drinking" example. The instantiated ontology model is shown in Figure 9.

- The single function at decomposition level 1 is mapped to a set of 5 sub-functions at level 2, using a **MapItem1N**. Since this is a downward mapping, its *role* is assigned the value 'realizedBy'.
- The input and output flows themselves are also identified as ***Mappable*** elements, and are associated via **MapItem11** individuals labeled e1 to e8. Note that these input and output flows actually denote the same things in levels 1 and 2 – they are neither realizations nor abstractions of each other. They are assigned the *role* of 'sameAs'.
- The mapping Mapping_0 from Level 1 to Level 2 then consists of these 9 **MapItem**s. For clarity, its *hasNodes* properties are implicitly shown as the light blue region, rather than drawing 9 edges.

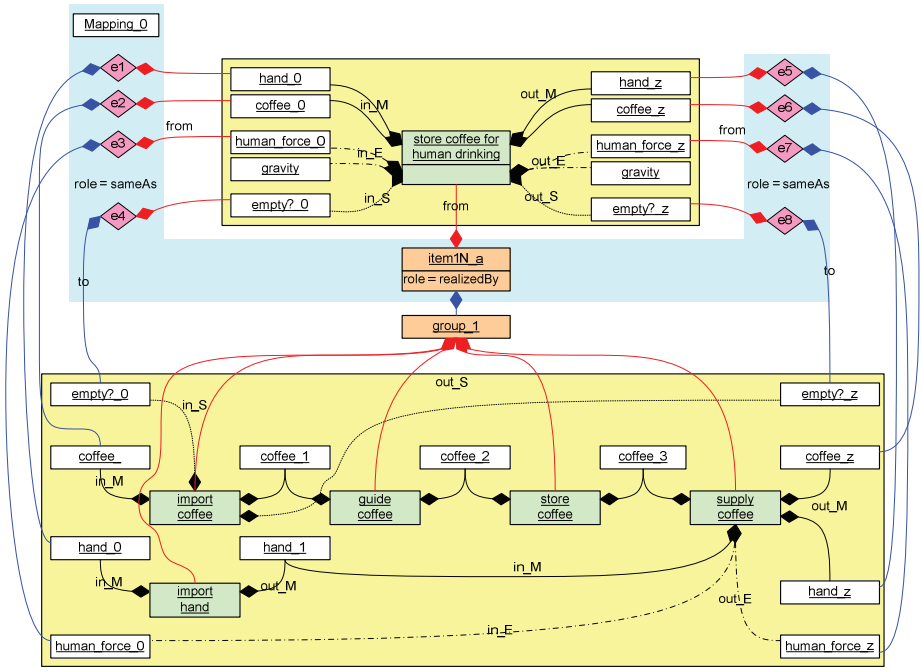

Figure 9. Mapping between Two Levels of Abstraction in a Function Decomposition

6 CONCLUSIONS AND FUTURE WORK

An ontology of PSS has been presented, with further elaboration on (1) the ontological representation of flow and functions, and (2) an ontology of mappings between two sets of elements. This representation of mappings has been applied between the three levels of values, functions, and structures, as well as between two levels of abstraction in a function decomposition.

Overall, the development of a comprehensive PSS ontology is a first step toward the eventual implementation of smart PSS design tools. The current PSS ontology is not intended to be directly used by human designers. But by incorporating established design methods such as function decomposition, it would support designers in using techniques and representations that are already familiar to them. And by building in the support for multiple levels of abstraction, this ontology could allow new perspectives of PSS designs to be visualized, e.g. to seamlessly go from a highly abstract (qualitative) view to a highly detailed (quantitative) view, and back, according to the designer's need.

Within a single level of abstraction, products could be decomposed into features and components, organized into a graph structure by connectivity and other relations [26][27]. We anticipate a need for a generic representation of decomposition knowledge, representing structural decomposition of an aggregate into its constituents, i.e. where the edges could include *part-of* relations. By defining a *part-of* role, we could extend our representation of mapping to also handle the decomposition of aggregates into constituents. This approach of extracting an ontological representation of generic decomposition knowledge, and reusing it across many domains, has also been used in Function and Behavior Representation Language (FBRL) [22].

In FBRL, the representation of flows (which are called *operands*) is extended with an explicit *state* parameter [22]. Recent research has explored the integration of function decomposition with notions of state and process [25]. We will extend our ontological representation to incorporate state and state change, as well as actors, and use these to extend our representation of function to encompass functions of services.

REFERENCES

[1] Tukker, A., and Tischner, U., "Product-Services as a Research Field: Past, Present and Future. Reflections from A Decade of Research", *J. Cleaner Production*, 2006, **14**, 1552–1556.

[2] Baines, T. S., Lightfoot, H. W., Evans, S., Neely, A., Greenough, R., Peppard, J., Roy, R., Shehab, E., Braganza, A., Tiwari, A., Alcock, J. R., Angus, J. P., Bastl, M., Cousens, A., Irving, P., Johnson, M., Kingston, J., Lockett, H., Martinez, V., Michele, P., Tranfield, D., Walton, I. M., and Wilson, H., "State-of-the-Art in Product-Service Systems", *Proc. IMechE, J. Engineering Manufacture*, 2007, **221,** 1543–1552.

[3] Goedkoop, M. J., van Halen, C. J. G., te Riele, H. R. M., and Rommens, P. J. M., *Product Service Systems: Ecological and Economic Basics*, 1999, Report for Dutch Ministries of Environment (VROM) and Economic Affairs (EZ).

[4] Mont, O., "Clarifying the Concept of Product-Service System", *J. Cleaner Production*, 2002, **10**, 237–245.

[5] Mont, O., *Product-Service Systems: Panacea or Myth?* Ph.D. Dissertation, Lund University, 2004.

[6] Smith, M. K., Welty, C., and McGuinness, D. L. (Eds), *OWL Web Ontology Language Guide*, W3C Recommendation, 2004, http://www.w3.org/TR/owl-guide/.

[7] Protégé, http://protege.stanford.edu/.

[8] Ohtomi, K., "Importance of Upstream Design in Product Development and Its Methodology", Keynote Presentation, *Proc. 6th IEEE EuroSimE Conf.*, Berlin, 2005.

[9] Tomiyama, T., Shimomura, Y., and Watanabe, K., "A Note on Service Design Methodology", *Proc. ASME Int'l. Conf. on Design Theory and Methodology*, Salt Lake City, 2004.

[10] Lindahl, M., Sundin, E., Sakao, T. and Shimomura, Y., "An Application of a Service Design Tool at a Global Warehouse Provider", *Proc. Int'l. Conf. on Engineering Design*, Melbourne, 2005.

[11] Sakao, T., Shimomura, Y., Comstock, and M., Sundin, E., "Service Engineering for Value Customization", *Proc. 3rd Int'l. World Congress on Mass Customization and Personalization (MCPC)*, Hong Kong, 2005.

[12] Sakao, T., Shimomura, Y., Comstock, M., and Sundin, E., "A Method of Value Customization", *Proc. Int'l. Design Conference*, Dubrovnik, 2006.

[13] Sakao, T., and Shimomura, Y., "Service Engineering: a Novel Engineering Discipline for Producers to Increase Value Combining Service and Product", *J. Cleaner Production*, 2007, **15**, 590–604.

[14] Kahle, L. R., Beatty, S. E., and Horner, P., "Alternative Measurement Approaches to Consumer Values: The List of Values (LOV) and Values and Life Style (VALS)", *J. Consumer Research*, 1986, **13**(3), 405–409.

[15] Maussang, N., Sakao, T., Zwolinski, P. and Brissaud, D., "A Model For Designing Product-Service Systems Using Functional Analysis and Agent Based Model", *Proc. Int'l. Conf. on Engineering Design*, Paris, 2007.

[16] Kim, Y. S., Wang, E., Lee, S. W., and Cho, Y. C., "A Product-Service System Representation and Its Application in a Concept Design Scenario", *Proc. CIRP Industrial Product-Service Systems (IPS2) Conf.*, Cranfield, UK, 2009.

[17] Pahl, G., and Beitz, W., *Engineering Design*, 1988 (Design Council, London).

[18] Kirschman, C. F. and Fadel, G. M., "Classifying Functions for Mechanical Design", *J. Mechanical Design*, 1998, **120**, 475–482.

[19] Stone, R. B., and Wood, K. L., "Development of a Functional Basis for Design", *Proc. ASME Conf. on Design Theory and Methodology*, Las Vegas, 1999.

[20] Gero, J. S. and Kannengiesser, U., "The Situated Function–Behaviour–Structure Framework", *Design Studies*, 2004, **25**, 373–391.

[21] Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., and Tomiyama, T., "Supporting Conceptual Design Based on the Function-Behavior-State Modeler", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1996, **10**, 275–288.

[22] van der Vegte, W., Kitamura, N., Koji, Y., and Mizoguchi, R., "Coping with Unintended Behavior of Users and Products: Ontological Modelling of Product Functionality and Use", *Proc. ASME Conf. on Computers and Information in Engineering*, DETC2004/CIE-57720, Salt Lake, 2004.

[23] Camelo, D., Mulet, E., and Vidal, R., "Function and Behaviour Representation for Supporting Flexible Exploration and Generation in a Functional Model for Conceptual Design", *Proc. Int'l. Conf. on Engineering Design (ICED)*, Paris, 2007.

[24] Nomaguchi, Y., Ohnuma, A., and Fujita, K., "Design Rationale Acquisition in Conceptual Design by Hierarchical Integration of Action, Model and Argumentation", *Proc. ASME Conf. on Computers and Information in Engineering*, DETC2004/CIE-57681, Salt Lake, 2004.

[25] Nagel, R. L., Stone, R. B., Hutcheson, R. S., McAdams, D. A., and Donndelinger, J. A., "Function Design Framework (FDF): Integrated Process and Function Modeling for Complex Systems", *Proc. ASME Computers and Information in Engineering Conf.*, DETC2008-49369, Brooklyn, 2008.

[26] Wang, E., and Kim, Y. S., "Form-Function Reasoning for Product Shape Ontology", *Int'l. Workshop on Semantic Web and Web 2.0 in Architectural, Product and Engineering Design, 6th Int'l. Semantic Web Conf. (ISWC)*, Busan, 2007.

[27] Wang, E., and Kim, Y. S., "Object Ontology with Justification Graphs for Form-Function Reasoning", *Proc. ASME Computers and Information in Engineering Conf.*, DETC2008-50016, Brooklyn, 2008.

Contact: Yong Se Kim
Creative Design Institute
Sungkyunkwan University
Suwon 440-746
Korea
Phone: +82-31-299-6581
Fax: +82-31-299-6582
Email: yskim@skku.edu
URL: http://cdi.skku.edu

Yong Se Kim is Director of the Creative Design Institute, and a Professor of Mechanical Engineering at Sungkyunkwan Univ., Korea. He received PhD in Mechanical Engr. from Design Division of Stanford in 1990. His research interest is Design Cognition and Informatics, which investigates fundamental processes in design, and provides methods and tools for design and design learning.

Eric Wang is a Senior Researcher in the Creative Design Institute at Sungkyunkwan University. His research interests include ontology modeling, rule-based reasoning, intelligent tutoring systems, geometric reasoning, feature-based computer-aided process planning, and systems development and implementation.

Myon Woong Park has been working on intelligent software for design and manufacturing since his Ph.D program at the University of Manchester which was completed in 1987. He currently serves as a principal research scientist of the Intelligence and Interaction Research Center and also a Professor for HCI and Intelligent Robotics at the Korea Institute of Science and Technology.