# Custom Digital Workflows: A New Framework for Design Analysis Integration

Bianca Toth[1], Patrick Janssen[2], Rudi Stouffs[2,3], Andre Chaszar[3] and Stefan Boeykens[4]

Queensland University of Technology
School of Design
2 George Street, D Block, Level 5
Brisbane QLD 4000 Australia
bianca.toth@qut.edu.au

National University of Singapore
Department of Architecture
4 Architecture Drive
Singapore 117566
patrick.janssen@nus.edu.sg, stouffs@nus.edu.sg

Delft University of Technology
Faculty of Architecture
Juliananlaan 134
2628 BL Delft, The Netherlands
a.t.chaszar@tudelft.nl, r.m.f.stouffs@tudelft.nl

KU Leuven
Department of Architecture, Urban Design and Regional Planning
Kasteelpark Arenberg 1 - box 2431
3001 Heverlee, Belgium
Stefan.Boeykens@asro.kuleuven.be

## Abstract

Flexible information exchange is critical to successful design-analysis integration, but current top-down, standards-based and model-oriented strategies impose restrictions that contradicts this flexibility. In this article we present a bottom-up, user-controlled and process-oriented approach to linking design and analysis applications that is more responsive to the varied needs of designers and design teams. Drawing on research into scientific workflows, we present a framework for integration that capitalises on advances in cloud computing to connect discrete tools via flexible and distributed process networks. We then discuss how a shared mapping process that is flexible and user friendly supports non-programmers in creating these custom connections. Adopting a services-oriented system architecture, we propose a web-based platform that enables data, semantics and models to be shared on the fly. We then discuss potential challenges and opportunities for its development as a flexible, visual, collaborative, scalable and open system.

# 1. INTRODUCTION

There is a clear and urgent need for better information exchange strategies to address the persistent lack of interoperability and integration in building design, analysis and construction. The continuing and active discourse amongst AEC practitioners and researchers alike highlights ongoing limitations in both process and technology that commonly challenge design collaboration. Recognising as a basic premise that most design teams will, whether by choice or necessity, use a variety of software applications and platforms, the question that remains to be answered is: How can we develop tools and technology that support designers in creating their own design processes, rather than having to adapt their processes to suit rigid tool requirements?

The key idea we present in this article is that bottom-up, user-controlled and process-oriented approaches to linking design and analysis applications are more appropriate than current top-down, standards-based and model-oriented strategies, because they provide degrees of flexibility critical to the process(es) of design. This idea originally comes from discussions raised at the "Open Systems and Methods for Collaborative BEM (Building Environment Modelling)" workshop held at the CAAD Futures 2011 Conference in Liège, Belgium. Here, we continue the 'open systems' dialogue with a conceptual framework for bringing this idea into practical application, aiming to reduce current obstructions to collaborative design.

We propose an open framework for integration where numerous small and specialised procedural tools are developed, adapted and linked ad-hoc, to meet the needs of individual design projects and project teams. These modular components encapsulate individual tasks that aid information exchange between software tools by (semi-)automating typically tedious and non value-adding tasks associated with preparing data for transformation between different data schemas. Flexible and user-friendly mapping procedures enable custom connections to be easily created and shared between schema descriptions adhering to the same underlying data model. A cloud-based platform allows various design and analysis tools to be strung together in this manner, through web interfaces that let users interact with workflows graphically. Project- and user-specific workflows can be created, shared and managed on distributed resources, which, in combination with the elimination of file format and programming language restrictions, promotes maximum flexibility.

Drawing on research into scientific workflows, we describe system requirements to guide future development of the proposed framework. In contrast with the predominating push for ever more monolithic and standardised tools, models and workflows, we dispense with the premise of a multi-domain schema for integration to present instead a system that is flexible, distributed and modular. Discarding assumptions of homogeneity in the format of design and analysis data, we then discuss the benefits and challenges that such a system presents for design practice and outcomes.

An initial demonstration of one possible approach to the shared mapping procedure has already been undertaken [1], and we have assembled a collective of researchers and practitioners interested in pursuing our proposal. We are confident that the approach described in this framework will lend itself well to coping with the frequently changing pace and focus of design projects, as well as the varying priorities of their many stakeholders.

## 2. SYSTEM ARCHITECTURE

Similar to the AEC sector, increasing complexity in scientific research and practice has led to a proliferation of specialised computational tools, each developed by different people, at different times, to support different problem-solving tasks. Across these tools, underlying data structures exhibit a high degree of heterogeneity, akin to that observed in building software. To manage this heterogeneity and achieve the integration required to generate solutions, information must be matched and mapped across a succession of different data schemas, applications and platforms [2]. Scientific workflows enable these information exchanges to take place quickly, reliably and flexibly, by "combining data and processes into a configurable, structured set of steps that implement semi-automated computational solutions of a scientific problem" [3].

Scientific workflow systems enable the composition and execution of these complex task sequences on distributed computing resources [4]. These systems exhibit a common reference architecture, illustrated in Figure 1, and typically consist of a graphical user interface (GUI) for authoring workflows (which can also be edited textually), along with a workflow engine that handles invocation of the applications required to run the solution [5]. The GUI abstracts the usage requirements of underlying resources to critical process parameters, making computational tools and technologies more accessible to users. The workflow engine supports integration between applications by engaging a combination of data-flow and control-flow

constructs to handle the execution and management of tasks. Data-flow constructs establish information dependencies between tasks, and ensure that data-producing procedures are complete before data-consuming ones begin [4]. Control-flow constructs support more complex workflow interactions, such as loops and conditionals, and also coordinate the execution of tasks on remote resources [5]. Typically, control-flow constructs are overlays on the data-flow graph, either as separate nodes or layers.
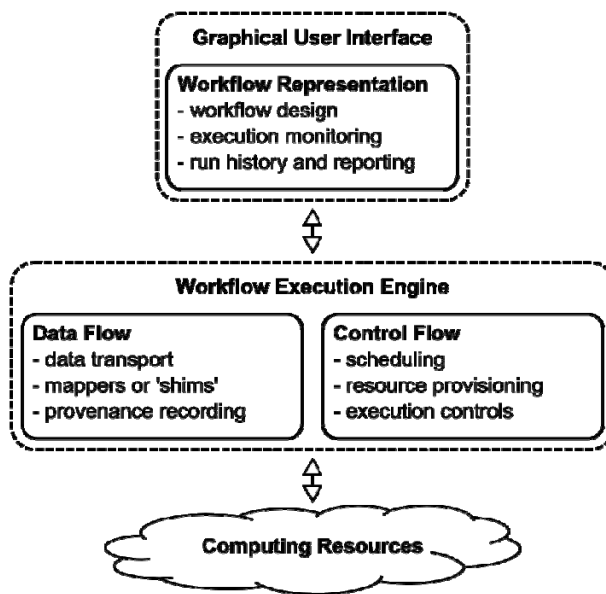


Figure 1: Workflow system architecture.

Today, numerous workflow systems with different purposes and functionality exist. Some provide sophisticated interfaces and graphics, like the data visualisation application Vistrails [6], while other more generic workflow systems, such as YAWL, are less visual but offer high-level process abstractions that can be applied to a range of usage scenarios [5]. The LONI Pipeline is designed specifically to build up data processing streams for neuro-imaging tools [7], while Kepler provides advanced control algorithms for actor-oriented modelling of complex physical, biological and chemical processes [5]. Each system acts to accelerate and streamline the workflow process; however, their individual capabilities vary greatly due to differences in workflow representation, data flow and control flow. The implementation strategies employed for each of these three workflow aspects reflect the specific requirements and technologies of the individual field or purpose for which a system is developed.

Within the AEC industry, systems developed for the purpose of Multidisciplinary Design Optimisation (MDO) offer comparable workflow functionality to those mentioned above. ModelCenter [8], by Phoenix Integration, is a commercial software package that integrates and manages the flow of data between modelling and simulation applications, to generate trade-off evaluations of design and performance constraints for decision support. Purpose-built software wrappers expose the data and functionality of design and analysis applications within the ModelCenter environment, enabling these tools to be linked together into workflows. DesignLink [9], an open-source platform by Arup, employs a different mechanism to achieve similar workflow functionality, supporting interoperability through the combination of an extensible XML-based file format and tool plug-ins for import and export of this format. While these MDO environments facilitate integration of diverse design and analysis tools, their top-down approach to interoperability is subject to certain limitations. Using of a common data schema to structure information exchange, regardless of whether it is proprietary or an open standard, imposes restrictions on how designs can be described and thus explored [10]; restrictions that are only lessened, not overcome, when the schema is extensible. Further complicating matters is the lack of support for non-programmers in creating custom software connections, given that software development expertise is relatively uncommon amongst architects and engineers [11]. In response to these observed limitations, we pursue a more flexible and user-friendly approach to information exchange that reconceptualises the workflow process to support the full scope of design experience, aiming to improve design-analysis integration across a range of application broader than simply MDO – including open-ended design explorations, not only optimisations. Our work differs from other related research in its bottom-up, rather than top-down, approach to integration, which eliminates the common data schema as a prerequisite for information exchange, allowing designers freedom to create custom digital workflows unfettered by standardisation constraints.
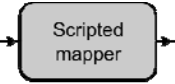
In the following subsections we discuss strategies for workflow representation, data flow and control flow in relation to the needs of the AEC industry, to outline requirements for a system that will support our bottom-up approach to information exchange. We aim to identify the functionality needed to develop a flexible, open and intuitive platform for collaborative design-analysis integration, building on capabilities exhibited in scientific workflow systems and further capitalising on recent advances in cloud computing.

## 2.1. Workflow representation

Workflow representation is critical for specifying tasks and dependencies. Nearly all workflow systems mentioned are visual programming tools in that they allow processes to be described graphically using some form of 'pipes-and-filters' logic. While not strictly workflow systems, programs like Grasshopper, GenerativeComponents, and Houdini abstract underlying CAD systems to offer similar functionality to designers for composing parametric-associative models, albeit performed within a single application. Each 'filter' encapsulates some data processing task, and is represented by a node, while a 'pipe' passes data (and in some instances control information) between two filters, and is represented by a connecting wire. A workflow is depicted by a network of nodes and wires to be configured and reconfigured graphically by users as required. From a user perspective, these nodes can act as a black box to perform a given function without the need for extensive or expert programming, although programming can empower the end user considerably, if the contents of the box are exposed and editable.

Adopting this 'pipes-and-filters' architecture, our framework posits three node types: process, input/output (IO) and control. Process nodes encapsulate data analysis and transformation procedures; while the latter two node types provide functionality related to workflow initiation, execution and completion. An overview of these different node types, and their respective sub-nodes, is seen in Table 1.

Table 1: Overview of workflow nodes.

| Node Type | Sub-node Type | Representation | Creation | Function |
|---|---|---|---|---|
| Process | Tool | Tool | Scripted | Wraps existing applications to make their data & functionality accessible to the workflow. |
| | Mapper | M | User-defined | Maps the output from one tool node to the input of another for simple transformations. |
| | Scripted Mapper | Scripted mapper | Scripted | Maps the output from one tool node to the input of another for complex transformations. |
| Input/Output | Input | input | System component | Provides data to the workflow by specifying input files and control parameters. |
| | Output | output | System component | Encapsulates workflow results and enables visualisation 'mashups' of these outputs. |
| Control | | control flow | System component | Applies constraints to the workflow that manipulate node execution, i.e. conditionals, loops. |

Process nodes have a number of (typed) input and output ports for receiving and transmitting data, as well as variables that can be set by the user to guide task execution. They can be further classified into 'tool' nodes and 'mapper' nodes. Tool nodes wrap existing applications to make their data and functionality accessible to the workflow, while mapper nodes apply transformation procedures to data sets to map the output from one tool node to the input of another. Ideally, these mappings are easily defined and able to be custom created (or edited) by users with limited programming skills (discussed in more detail in Section 3).

However, the need for more complex data transformations is inevitable, and where programming is unavoidable, the resulting nodes are differentiated as scripted mappers. Figure 2 shows an example network in which a Maya modelling node is connected via a series of mapper nodes (denoted by 'M') to Radiance and EnergyPlus simulation nodes. The Maya node encapsulates a procedure that starts Maya, loads a specified model, and then generates a model instance by applying the defined parameter values. The resulting geometric output undergoes two separate transformations that map it into both Radiance- and EnergyPlus-compatible formats, the latter via a scripted mapper that pre-processes geometry to ensure its compatibility with complex EnergyPlus requirements. The simulation nodes then read in this transformed data, run their respective simulations, and generate output data in the form of simulation results.
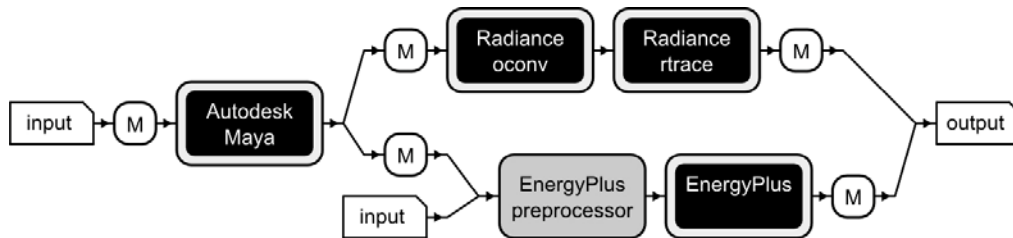


Figure 2: Example of a workflow with a parametric CAD system linked to Radiance and EnergyPlus via mapper nodes (M) – end users contribute white components, node developers build the grey components, and black components are existing tools that are wrapped.

IO nodes act as data sources and sinks for the workflow, as well as providing data visualisation capabilities. Input nodes provide data by specifying input files and control parameters, as inputs to the tool nodes and to control the data extracted from these input files respectively. It is important to note that input data is referenced into the workflow, not imported, to ensure that the workflow remains dynamic and can be re-evaluated when changes to this data occur; these changes are, however, controlled to avoid conflicts arising while the workflow is running. Taking the example in Figure 2, the Maya input node allows the user to specify not only the origin of the model to be used for the data source, but also particular types of geometry and other data contained within it, while the EnergyPlus input node simply links the appropriate weather file. The output node contains the workflow results, here of the EnergyPlus and Radiance simulations.

These nodes enable results to be visualised in various ways, with users able to define data 'mashups' in order to customise their visualisations without having to understand the coding of the underlying processes. Control nodes apply constraints to the workflow, like conditionals and loops, which manipulate the local order of execution of nodes further along in the network. For example, an if-then node can force execution of different branches in a workflow in response to a given condition, while a repeat node can force repeated execution of a network branch a particular number of times. Global control is also possible, but is defined at workflow level, rather than task level, as discussed in Section 2.3.

Users configure nodes and their dependencies using a workflow interface. Since we are describing a platform that operates in the cloud, this interface must be a web application able to access distributed cloud services. We propose a browser-based GUI that employs HTML5 to ensure its usability in standalone applications, or even smartphone apps. This GUI provides drag-and-drop functionality for placing nodes on the workflow canvas, which are then wired together by the user, similar to defining a model in Grasshopper, or in the various scientific workflow platforms mentioned above.

We also propose several interface features to aid users in managing workflow complexity. Graphical nesting allows clusters of nodes to be collapsed into composite nodes, facilitating modularisation of the workflow to improve its legibility [12], as well as enhancing reusability through the sharing of these clusters amongst workflows. Provenance information retrieval and querying enables workflow history to be reviewed, so that the decision-making process can be tracked [4]. Further functionality includes the ability to download the source code for any non-proprietary scripted node, modify it and then republish it. This is crucial for ensuring the platform's flexibility and openness; however, it does require a clear understanding of how data is passed through the system, discussed in more detail below.

## 2.2. Data flow

Interoperability is a critical issue when linking applications from different domains. Scientific workflow systems deal with this in a number of ways, ranging from an ontological approach, where a common data schema is imposed on data exchanges, to an open-world approach, where the user resolves data compatibility issues as needed - a process known as 'shimming' [3]. In the AEC industry, a common approach to this issue is Building Information Modelling (BIM), which tends toward the all-encompassing ontological end of the spectrum. This is a top-down approach, reliant on the IFC schema and its continuous

extension to cater for all possible usage scenarios. There, pragmatic information exchange is assumed to evolve into discipline-oriented model views, where only filtered subsets of the model are exchanged [13]. However, IFC – like any standardised ontology – has significant epistemological, practical and technical limitations [10, 14, 15]. With a focus on data specification, rather than on implementation of this specification, the data content involved in IFC exchange is often ambiguous and inconsistent between software [15]. There is also a lack of agreement concerning IFC model view definitions, which often results in file exchanges involving large, cumbersome models unable to resolve differences in data structure and content reliably between applications [14]. Furthermore, rather than actively supporting variability in design processes, usage scenarios are limited to that which is accommodated by the current IFC implementation (or risky 'hacks' thereof).

Instead of reading and writing to a common data schema, we therefore propose tools be coupled more effectively through procedures that allow direct data transfer between schemas. As advocated by proponents of design-analysis integration, this transfer only needs to involve the required data rather than the entire building description [16]. The advantage of this run-time coupling of different applications, Hensen et al. argue, is that it "supports the exchange of information during a simulation as opposed to ... before the simulation" [17]. While this approach is vulnerable to version changes in wrapped tools, the sharing and reuse of interoperability solutions would be a mitigating factor. Furthermore, the proposed system does not disregard BIM, but suggests IFC exchange be part of the workflow process, integrated into these custom data flows rather than forcing the whole system to adhere to IFC ontology. A good example of such an approach is found in the "GeometryGym" suite of tools, which enable parametric models generated in Grasshopper to be linked both to BIM workflows, through components that generate IFC objects, and to structural simulations [18].

The absence of any schema restrictions circumvents the limitations of a multi-domain schema like IFC, where exchanges between tools and the underlying data schema pose risks of significant information loss and inappropriate process constraint. The only necessary constraint we envision is that domain-specific input and output data schemas are described and their assumptions identified, such that users may reasonably rely on these descriptions when creating and editing workflows. This constraint does not limit flexibility, as a new schema may always be defined.

Data can be stored in a digital repository, or a key-value database, a selection that is prompted as much by the elimination of any data schema restrictions as by the choice of a cloud-based platform. The advantages of these are that they do not impose a fixed schema for data description, are highly scalable, and information access can be very fast. The key for a piece of data is a unique ID, and its value can be a file, a class, a graph or any other data type. It should be noted that the use of discrete packages of data over a centralised process model may introduce some redundancy and inconsistency; for example, when different workflow branches drawing on the same input converge, combining data from different but overlapping models. Such redundancy, admittedly, is inherent to the bottom-up approach to integration that we are advocating. Eliminating this redundancy, however, would not only greatly reduce the freedom and flexibility of designers to create their own workflow processes from any selection of tools, it would also seriously hamper the ability to define and explore unconventional design spaces.

## 2.3. Control flow

As discussed in Section 2.1, control nodes provide localised ways of manipulating the workflow. To provide the desired level of flexibility, the framework also needs to offer different types of global, or high-level, control flow. Many existing workflow systems are restricted to simple flow mechanisms, where nodes are processed one at a time, in a pre-calculated order. Such systems generate topological orderings, where each node in the sequence will execute only after all its predecessor nodes have executed. A key limitation of this approach though is that the network must be a directed acyclic graph (DAG) to generate topological orderings, and as a result networks with loops are not supported.

Networks with loops, however, are clearly desirable in certain situations, such as optimisation routines. To support loops and other node execution patterns, such as triggering nodes iteratively or periodically, different high-level control mechanisms are required. In addition to providing workflow execution functionality, these mechanisms are needed to support distributed computing, by triggering nodes to work in parallel with other nodes, as well as executing synchronously or asynchronously.

To ensure maximum flexibility, the user should be able to apply different control flow mechanisms to different parts of the network. This could be achieved by assigning a control mechanism to a composite node, which would further open the possibility of nesting control flow.

# 3. CUSTOM DATA MAPPING

In order to create custom workflows, information must be matched and mapped across a series of different applications and schemas. This may involve deriving new data as well as disregarding data that is superfluous in the target format. An explicit model describing these transformations is therefore needed to relate constructs in the source representation to constructs in the target representation [2, 19].

In a workflow system that dispenses with file format and programming language restrictions, like ours, these transformations are a critical issue for usability. To support a bottom-up, user-controlled and process-oriented approach to linking design and analysis tools, a common mapping approach is needed that allows users with limited programming skills to easily create and share custom mappings, so that data compatibility issues can be resolved collaboratively on the fly. For this to be achieved, the mapping approach must be both *flexible*, so that users can apply the same mapping approach to any type of data, and *user-friendly*, to the extent that it supports non-programmers in the process of creating and debugging mappers. This approach must therefore be semantically agnostic, employing data models that support the organisation of information at an abstract level, in terms of the data structure, data constraints and data operations. By making the output schema of one node and input schema of another adhere to the same data model, a mapping between two tools may then be more easily defined.

## 3.1.   A flexible mapping approach

The mapping process involves two distinct levels of data representation: the *data model* versus the *data schema*. A data model allows data to be organised using generic constructs that are domain independent and able to describe a broad range of information types. It is a means of defining a data structure that is very flexible but relies on human interpretation of semantic meaning. A data schema describes information using semantic constructs related to a particular domain. While the range of information types supported is typically narrower, this level of representation allows automated interpretation of semantic meaning. We propose a mapping approach that engages both of these representation levels in order to ensure that it is flexible and capable of automatically generating transformation procedures from abstract mapping definitions.

Data schemas can be defined on top of a data model by specifying additional constraints for the underlying generic constructs. One example is the different XML schemas defined on top of the XML data model

using the XML Schema DDL. In this manner, data in any format can be made to adhere to the same data model, easing the mapping definition process by ensuring commonality in the way that information is represented between schemas. Figure 3 illustrates this layered approach (option c) in comparison with other AEC strategies for interoperability. We can see here that its advantage lies in providing a consistent method for defining tool connections, which is more cohesive than linking individual tools via file translation (option a), and less restrictive than a universal data schema (option b).
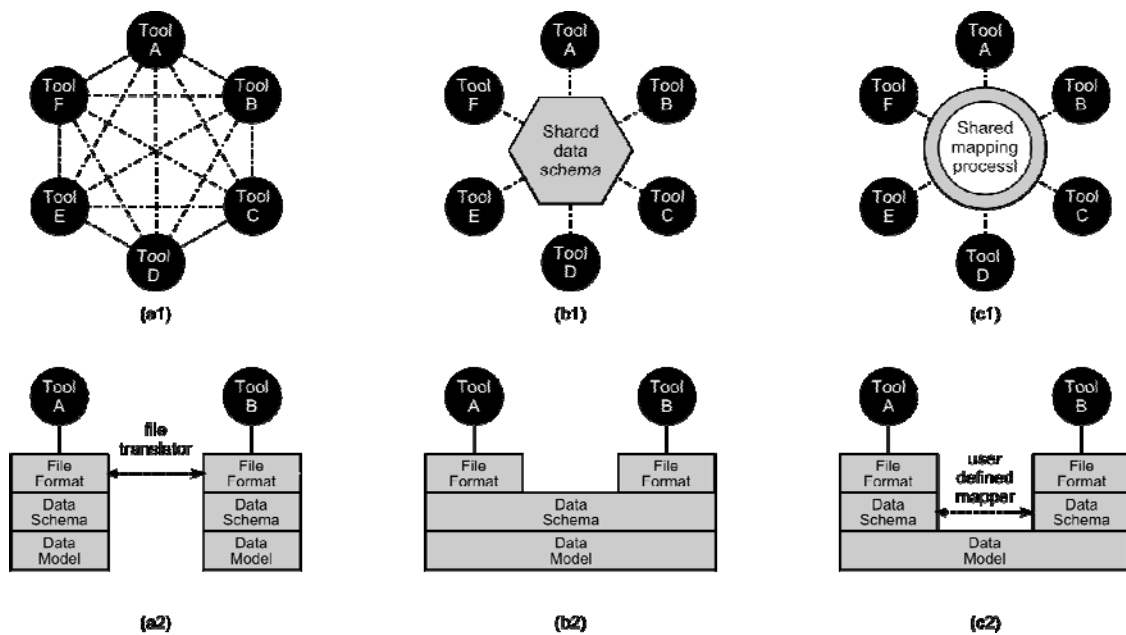


Figure 3: Interoperability approaches. Left: Linking individual tools via file translation. Middle: Linking tools via a shared data schema. Right: Linking tools via a shared mapping process.

Mappings typically need to manipulate both data structure and content to generate the required exchange procedure. This is achieved by specifying relationships between domain-specific constructs in the output schema and those in the input schema. In the modelling and simulation of design problems, the resulting relationship networks are often highly complex, and cannot be defined through a simple declaration of semantically-equivalent constructs between the two schemas. To minimise the need for scripted mappers in creating such networks, the mapping approach must allow complex relationships to be defined in a highly flexible and intuitive manner. We propose to support this flexibility by enabling schema relationships to be

described through transformation rules, using languages designed specifically for querying and manipulating data models. In keeping with the data models that they are used to traverse, these languages are highly generic semantically, and offer the flexibility needed to negotiate relationships between disparate design and analysis schemas. It should be noted, however, that each is unique to a particular type of data model. In a further paper, we have considered several data models and their respective query languages from the point of view of applicability and ease of use [1]. Since adherence to a common data model is only a prerequisite for nodes in any individual mapping, to ensure maximum flexibility, we propose to dispense with data model restrictions at the system level.

## 3.2.    A user-friendly mapping approach

Similar to other workflow systems, to assist the designer in creating these custom mappings, we propose a mapping interface that supports visual definition of transformation rules. In our system, the input and output data schemas of each tool and scripted mapper node are represented as a graphical abstraction based on a data model (such as a property graph). Using graphical constructs that represent high-level transformations, designers are able to visually define relationships between the output and input schemas of different nodes to generate a full specification of the data exchange procedure [2]. Based on this user-defined information, a mapping procedure that transforms the source data set into the target data set can then be generated automatically. Ideally, this interface would employ advanced algorithms that offer guidance to the designer in matching data across schemas, so that mappings can be created in less time, with less effort and fewer mistakes [2]. An example of this can be seen with elastic lists [20], where the selection of one object for mapping places logical limitations on the properties that can be selected thereafter. This would allow users to create transformation procedures without any programming, although the interface would also enable direct editing of the transformation script so that procedures could be enhanced for maximum flexibility.

For tools structuring their input and output schemas using this same data model, the resulting commonality would allow these tools to be easily shared. Users could download diverse tools (and possibly mappers) developed by different groups from a shared online repository, and then string these together into their own unique customised workflows. The users' task would then be to define (i.e. create or edit) mappers, where necessary, that generate data sets adhering to the input schemas of the selected tools.

# 4. SYSTEM IMPLEMENTATION

Implementing our framework in the cloud promotes its scalability, efficiency and reliability, as workflow execution can be distributed over multiple computers in the network. Nodes may execute either as local tasks, if their related design or analysis program resides on the local machine, or as cloud applications, if their data-handling procedures are not linked to a locally-housed program, and they are able to swap between modes of execution as required. The node procedure, which may be written in any language, is saved as an executable task that reads and writes data. A task scheduler, housed on a server, is used to manage the execution of these tasks on distributed resources, and the user controls the execution process through a web-based workflow GUI that communicates with this scheduler. If the system needs to be run locally, such as when a network connection cannot be established, a server is simply started on the local machine.

Process nodes read and write data to and from a (distributed) repository. Storing data in a repository ensures that a trace of all workflow output is maintained for later perusal. The repository may be mirrored locally, guaranteeing access to all outputs even when the user is disconnected from the cloud. Similar to version-control systems used in software development, such as Subversion or Git, users work on a local copy and then merge or commit their changes to the shared repository once the workflow has been executed. Data management within a repository also eliminates the need for exchanging data directly between process tasks. When a task creates output data, it is stored in the repository together with some minimal metadata - origin, time of creation, format, etc. - for which the Dublin Core [21], extended where necessary, may serve as a template. In return, it receives a unique ID, which is then passed to the task scheduler to be forwarded on to the next process node's task. The receiving task then queries the repository for the relevant data.

Literature on scientific workflows also suggests several other desirable system features [3], including:

- reusable features that are both generic and customisable to support a wide range of tasks in specific ways;
- seamless and dynamic subscription and publishing to data and processes from repositories;
- execution monitoring, failure recovery and interactive user guidance of workflows.

This functionality will be incorporated into the future development of the system, as well as steer the implementation of features already discussed.

## 5. DISCUSSION

The question of interoperability has long been vexing the AEC industry. While computational systems for design-analysis integration, for example PACE [22], SEMPER [23] and P3 [24], have been the subject of research for decades and instrumental in establishing a foundation for current integration efforts, they have largely not been adopted in practice. Their inability to support the *ad hoc* workflows experienced on real projects [25] is a problem similarly observed in integration practices today, both of which can be attributed to implementation of an interoperability approach based on a common data schema. By imposing inflexible standards for data representation and exchange, restrictions are placed on the software and processes that can be supported, and severe limitations are inflicted on the ways in which designs can be described and explored. To overcome these limitations, we are proposing that direct communication between different data schemas facilitated by an open mapping approach - rather than communication via a multi-domain schema - is the preferred and even necessary arrangement. Although more work may be needed to create the many tool wrappers and scripted mappers required to support such communication, this is more desirable than over-constraint of design processes caused by the use of standardised representations and protocols that are only applicable within a relatively small region of 'design space'. These nodes do not need to be created in advance, but can be developed or adapted by designers as the need arises, and shared amongst the user community. This promotes greater responsiveness to user needs than when software development is left in the hands of firms motivated by commercial considerations.

Research and development of the proposed system is an ongoing effort from the Open Systems group; however, success will ultimately depend on a community of users and developers, able and willing to adapt this system to their own workflows, capitalising on tools and processes they already have in place. We do not aim to replace their toolsets but rather provide a framework to embed current tools in more cohesive, shareable and customisable digital workflows. As observed within the Grasshopper user community, the ability to influence and effect software extension, and benefits of combining various add-ons to functionality, encourages users to contribute to system development. Consultants can also showcase their skills by sharing select process nodes that they develop in-house, as is similarly achieved by the

GeometryGym and Karamba plug-ins for Grasshopper, to establish themselves as experts to prospective clients and co-collaborators who require specialised workflows for unique design projects and problems. These user developments could potentially result in a large collection of process nodes, which may compete in offering alternative ways of achieving almost identical outcomes, and a considerable range of options available to designers. To aid designers in choosing appropriately, it is important not only that a description of the functionality of each node is available, but also that the designer is able to ensure that nodes 'fit' other nodes in order to compose a valid workflow. Formal input and output schemas for tool nodes should also be made readily available, to assist users in determining which subsets of these schemas are relevant to their particular design scenario. Assertions should also be specified on data outputs and assumptions specified for expected inputs so that automatic checking of node compatibility is possible, and where compatibility is lacking a new or modified node is substituted.

Workflow design is likely to be an incremental process in which a number of nodes are combined into a partial workflow, tested by the designer, then further developed and extended. Besides automatic checking of the mutual fitness of adjacent nodes, the designer will need to check whether the (partial) workflow is behaving as expected and producing appropriate results. When results are not as expected, the designer will need to debug the workflow by tracing back execution, which can be assisted by displaying intermediate, as well final, results.

In the context of a large collection of process nodes and choice of alternative ways to achieve the same or similar result, user-friendliness and knowledge-based support, the two main concerns of designers when using analysis software [26], will become crucial. These concerns will also need be explored in terms of the scalability of the user-defined mapping approach when working with large data sets and more complex data schemas (like IFC). If the increased complexity makes it difficult to construct custom mappers, then additional data management and schema management tools will be required – tools which could also assist in workflow debugging and schema formalisation. Issues of accuracy, uncertainty and risk may also be of significant concern. Macdonald et al. propose the introduction of uncertainty considerations in simulations to provide meaningful feedback to the user and to improve confidence through risk assessment [27]. While these can be addressed within individual software tools, the proposed system should also introduce this functionality into the workflow environment itself.

A final issue is security and trust. In a collaborative and distributed environment, how can we guarantee that data is secure and processes can be trusted? Data encryption and certification of process nodes will be prerequisites, in combination with a secure workflow platform, to ensure trust to the user.

## 6. CONCLUSION

This article summarises an ongoing effort to address limitations in process and technology that currently obstruct design collaboration. In it we argued the need for a user-controlled and process-oriented approach to integration and interoperability, and discussed how a cloud-based workflow system can support more flexible and distributed design processes. We examined the features and functionality needed to abstract computing and data resources to make tools and technologies more accessible to users, both as individuals and as members of design teams. As well as benefiting design practice, we envisage the proposed system as a platform for researchers to share their work and increase the impact of their individual efforts through integration with other research. The system requirements that we have identified will help to ensure that the proposed design-analysis integration platform when developed is flexible, visual, collaborative, scalable and open.

## Acknowledgements

## References

1. Janssen, P., Stouffs, R., Chaszar, A., Boeykens, S. and Toth, B., Data Transformations in Custom Digital Workflows: Property Graphs as a Data Model for User-Defined Mappings, Intelligent Computing in Engineering Conference - ICE2012, Munich, 2012.

2. Bellahsene, Z., Bonifati, A., Duchateau, F. and Velegrakis, Y., On Evaluating Schema Matching and Mapping, in: Bellahsene, Z., Bonifati A. and Rahm E., eds., Schema Matching and Mapping, Springer, Berlin and Heidelberg, 2011, 253-291.

3. Altıntaş, İ., Collaborative Provenance for Workflow-driven Science and Engineering, PhD Thesis, University of Amsterdam, 2011.

4. Deelman, E., Gannon, D., Shields, M. and Taylor, I., Workflows and e-Science: An Overview of Workflow System Features and Capabilities, Future Generation Computer Systems, 2008, 25, 528-540.

5. Curcin, V. and Ghanem, M., Scientific Workflow Systems - Can One Size Fit All?, CIBEC 2008, Cairo, 2008, 1-9.

6. Callahan, S., Freire, J., Santos, E., Scheidegger, C., Silva, C. and Vo, H., Vistrails: Visualization Meets Data Management, SIGMOD 2006, Chicago, 2006, 745-747.

7. Rex, D., Ma, J. and Toga, A., The LONI Pipeline Processing Environment, Neuroimage, 2003, 19(3), 1033–1048.

8. http://www.phoenix-int.com/software/phx-modelcenter.php [30-10-2012]

9. http://arupforge.arup.com/wiki/index.php?title=DesignLink_SDK [30-10-2012]

10. Bowker, G. and Starr, S., Sorting Things Out: Classification and Its Consequences, MIT Press, Cambridge, 1999.

11. Flager, F., Welle, B., Bansal, P., Soremekun, G. and Haymaker, J., Multidisciplinary Process Integration and Design Optimization of a Classroom Building, ITcon, 2009, 14, 595-612.

12. Davis, D., Burry, J. and Burry, M., Untangling Parametric Schemata: Enhancing Collaboration Through Modular Programming, CAAD Futures 2011, Liège, 2011, 55-68.

13. Eastman, C., Teichholz, P., Sacks, R. and Liston, K., BIM Handbook - A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors, 2nd edn., John Wiley and Sons, Hoboken, 2011.

14. Kiviniemi, A., Ten Years of IFC Development - Why We Are Not There Yet, Proceedings CIB-W78, Montreal, 2006.

15. Pazlar, T. and Turk, Z., Interoperability in Practice: Geometric Data Exchange Using the IFC Standard, ITcon, 2008, 13, 362-380.

16. Augenbroe, G., deWilde, P., Moon, H., Malkawi, A., Brahme, R. and Choudhary, R., The Design Analysis Integration (DAI) Initiative, 8th IBPSA Conference, Eindhoven, 2003, 79-86.

17. Hensen, J., Djunaedy, E., Radošević, M. and Yahiaoui, A., Building Performance Simulation for Better Design: Some Issues and Solutions, PLEA 2004, Eindhoven, 2004, vol. 2, 1185-1190.

18. Mirtschin, J., Engaging Generative BIM Workflows, Collaborative Design of Lightweight Structures - LSAA 2011, Sydney, 2011, 1-8.

19. Kilian, A., Design Innovation Through Constraint Modeling, International Journal of Architectural Computing, 2006, 4(1), 87–105.

20. http://well-formed-data.net/experiments/elastic_lists/ [30-10-2012].

21. http://dublincore.org [30-10-2012]

22. Maver, T., PACE 1: Computer Aided Design Appraisal, Architects Journal, 1971, July, 207-214.

23. Mahdavi, A., Mathew, P., Lee, S., Brahme, R., Kumar, S., Liu, G., Ries, R. and Wong, N., On the Structure and Elements of SEMPER, ACADIA '96, Tucson, 1996, 71-84.

24. Kalay, Y., P3: Computational Environment to Support Design Collaboration, Automation in Construction, 8(1), 37-48.

25. Holzer, D., Optioneering in Collaborative Design Practice, <u>International Journal of Architectural Computing</u>, 2010, 8(2), 165-182.

26. Attia, S., Beltrán L., De Herde, A. and Hensen, J., "Architect Friendly": A Comparison of Ten Different Building Performance Simulation Tools, <u>11th IBPSA Conference</u>, Glasgow, 2009, 204-211.

27. MacDonald, I., Clarke, J. and Strachan, P., Assessing Uncertainty in Building Simulation, <u>6th IBPSA Conference</u>, Kyoto, 1999, vol. II, 683-690.