MODELLING COMPLEX AND FLEXIBLE PROCESSES FOR SMART CYBER-PHYSICAL ENVIRONMENTS

Ronny Seiger[(a)], Christine Keller[(a)], Florian Niebling[(a)], Thomas Schlegel[(a)]

[(a)] Junior Professorship in Software Engineering of Ubiquitous Systems, Technische Universität Dresden, Germany

[(a)] {ronny.seiger, christine.keller, florian.niebling, thomas.schlegel}@tu-dresden.de

ABSTRACT

Cyber-physical Systems introduce several new requirements for modelling and executing autonomous processes. Current workflow languages are not able to completely fulfil these requirements, as they mostly lack expressiveness and flexibility. In this paper, we therefore present a new workflow language for formalizing processes within heterogeneous and dynamic environments. Our approach is highly model-based and uses aspects of component-based software engineering. We present an object-oriented meta-model for describing processes, which enables the hierarchical composition of process components and leverages reusability. In addition, a domain-specific model is used for typification of process elements. Due to the object-orientation, we are able to easily extend our models and create variants of processes. Type-based modelling and polymorphism enable the dynamic selection of appropriate process steps at runtime, creating flexible processes. We present a graphical editor and a distributed execution engine for our meta-model. In addition, we discuss the use of semantic technologies for smart workflows.

Keywords: workflow language, process modelling, cyber-physical systems, meta-modelling, smart factory, automation

1. INTRODUCTION

Business processes have gained an increasing importance in describing complex correlations between distributed systems and executing composite workflows. Especially in the field of online trading and manufacturing, modelling and execution languages for business processes, e. g, BPMN and BPEL, have proven to be well suited to formalize high-level sequences of tasks and activities involving web service invokes and human interaction.

However, the on-going integration and combination of embedded systems and distributed cloud-based services into cyber-physical systems (CPS) and smart environments, lead to a number of new requirements for process modelling and execution. Most current workflow languages lack structure, expressiveness, and flexibility to meet these requirements.

Some of the drawbacks of state of the art process modelling languages include: only weak means for typing of process components and data, mostly static calls to a fixed set of service types, and reduced flexibility considering runtime modelling and adaptation. Modelling tools often produce code, which is incompatible with execution environments, and only a subset of the model elements is supported.

In addition, many long-established workflow modelling languages have been extended and evolved over time, mostly by adding new components and modifying the respective meta-models in order to meet new requirements and provide new functionality. This has led to complex and ambiguous process modelling languages containing special solutions for specific problems and domains.

In this paper, we present a new meta-model for processes designed to meet the requirements of current and future ubiquitous systems. We believe that by using model-based approaches, we can create a modular and extensible workflow language. With the help of this language, we will then be able to model flexible and dynamic processes for the automation of workflows. Current semantic technologies will help us with developing a smart and context-adaptive process engine and modelling environment. We focus on adhering to simple structures for the core of the process meta-model and at the same time being able to easily extend this model by means of object-orientation. Nevertheless, we are able to map process models of other workflow languages to models compatible with our system.

The paper is structured as follows: Section 2 presents some basic terms and explanations. Section 3 lists requirements that are introduced with the emergence of ubiquitous systems. Section 4 gives a brief overview of related work and evaluates state-of-the-art workflow languages with respect to their suitability for cyber-physical systems. Section 5 describes our own process model for complex and flexible business processes in detail. Section 6 demonstrates practical aspects with respect to implementing the model, a modelling tool, and a process execution engine. Section 7 discusses our approach and shows some aspects to further extend our research. Section 8 concludes the paper.

2. BASIC CONCEPTS

We will start with clarifying basic terms and concepts that are used within the context of this paper. As our focus lies on the scope of ubiquitous computing and cyber-physical systems, we will introduce these concepts first, as well as, our understanding of processes. Second, the paradigms of model-driven architecture and meta-modelling will be presented, as our own approach is based on these concepts.

2.1. Ubiquitous Computing

In his article "The Computer for the 21s Century", published in 1991 (Weiser, 1991), Mark Weiser introduced his vision of "the age of calm technology, when technology recedes into the background of our lives" and thereby coined the term "ubiquitous computing". Ubiquitous computing can be found at the intersection of pervasive computing, mobile computing, and ambient intelligence, and stands for systems that are unobtrusively integrated into everyday objects and activities.

2.2. Cyber-physical Systems

Cyber-physical systems (CPS) can be regarded as a major step towards Weiser's vision. CPS comprise networks of embedded, heterogeneous sensors and actuators into complex distributed systems, that are often linked to cloud-based services and cross-boundary systems. A closed loop between local sensing, remote processing, and local controlling can often be found within cyber-physical systems. Real-world objects are represented digitally and taken into consideration when planning and executing processes in a cyber-physical system. In addition, CPS are highly dynamic with respect to their components, i.e. devices and services can be added and removed at any time. By constantly collecting context information (Abowd et al., 1999), cyber-physical systems are able to adapt themselves to the current users and environment, thus evolving into so-called "smart spaces", e.g. smart homes, smart offices, and smart factories. CPS intend to create a strong link between the physical world and the cyber world, and to support their users with performing their daily tasks.

2.3. Processes

Processes (workflows) have been used to describe complex sequences of tasks and function calls in order to model the high-level behaviour of so-called systems of systems. Due to the large increase of distributed and loosely coupled systems over the last decades, the need for an additional layer describing workflows between multiple entities has been generated. With traditional approaches, it is not possible any more to implement all algorithms and cross-boundary interactions within the software application shipped with one product. The usage of processes helps with creating autonomous environment and the automation of repeating tasks.

We therefore define a process for the scope of our work as follows:

Processes represent a set of actions (process steps), which are connected with each other by a unidirectional order relation describing the order of execution of the steps (Schlegel, 2008).

2.4. Model-driven Architecture & Meta-Modelling

Using models throughout the development process of a software system incorporates several advantages with respect to modularization, reusability, extensibility, automatic code generation, and maintenance. The process layer on top of software products and systems should also be highly model-based and described by a platform independent model (PIM).

With the Meta-Object Facility (MOF), the OMG (http://www.omg.org/mof/) introduced a de facto standard for model-driven engineering (Aßmann et al., 2006)), describing several (meta-) levels of abstraction for modelling various kinds of systems. As we will also be dealing with models and meta-models throughout this paper, we want to clarify our understanding of these terms and their use within the context of process modelling at this point.

- *Process Meta-Meta-Model:* A process meta-meta-model (MOF-M2) defines the semantic and syntactic elements and structures used in the process meta-model.
- *Process Meta-Model:* A process meta-model defines all elements, types, and relations that can be used for modelling processes as well as their structural combinations. The process meta-model (MOF-M1) is an instance of the process meta-meta-model.
- *Process Model:* A process model is the abstract description of an actual process, which can be instantiated and executed at runtime. The process model (MOF-M0) is an instance of the process meta-model.
- *Process Instance:* A process instance represents a concrete process at execution time, having a runtime state. The process instance is an instance of the process model.

In the main part of this work, we will put our focus on presenting a new process meta-model, but we will also briefly describe the underlying process meta-meta-model.

3. REQUIREMENTS FOR MODELLING UBIQUITOUS PROCESSES

In order to evaluate current workflow languages with respect to their suitability for being used within ubiquitous systems (UbiSys), we will first outline some special requirements that come along with developing ubiquitous systems. Some of the following requirements are already predominant within current system architectures. However, UbiSys combine them to a large degree.

- **Dynamics:** Ubiquitous systems, as well as cyber-physical systems (CPS), are characterized as being highly dynamic with respect to the number and availability of its components, devices, and services. Therefore, modelling service invocations within processes on the instance level, i.e. the invocation of a concrete service, may not be suitable due to its possible unavailability. Hence, we also need to be able to model process steps and service calls on the type level, i.e. a certain type of service should be invoked. This way, we do not necessarily need to know at modelling time, which concrete service or device will be executing the process step.
- **Heterogeneity:** In a CPS there are usually numerous heterogeneous services and devices integrated into a so-called system of systems. However, when modelling workflows, a unified view on these components would be helpful. In addition, we would like to support a wide range of different services types and be able to easily extend this set. Complementary to the aforementioned requirement, there should also be a way of assigning an activity to a certain handling entity (resource) on the instance or the type level.
- **Complexity:** Processes within CPS can be very complex and contain a large number of process steps, both, composite and atomic, as well as further process elements. This makes means for hierarchical structuring and aggregating process components necessary in order to master high levels of complexity. A modular meta-model can also leverage exchangeability and reusability of process components.
- **Parallelism:** Numerous process instances may exist in parallel in a CPS and their execution times and cycles can also vary considerably. As processes often influence other processes indirectly, i.e. without an explicit specification of the interrelations within the process model, there should be means for supporting this way of process interaction and intercommunication available. This will leverage the integration of loosely-coupled systems and processes.
- **Evolution:** **W**ith respect to long-lasting processes, there may also be a need for changing the underlying process model during instance execution, due to a change of conditions or within the context of the process environment. It is often necessary to generate variants of models in line with new requirements or needs, e.g. within custom industrial production processes.
- **Distribution:** An important additional aspect concerns the execution of the process models. Engines for executing business processes are usually designed to be a central orchestration entity calling the services corresponding to the business process model. However, in a cyber-physical environment, e.g. a smart home, there often is no central high-performance server available. Instead, several small low-powered devices are distributed and embedded into the environment. Our future aim is to also use these resources for executing parts of a process in a distributed way.

4. RELATED WORK

During the last years, a lot of special purpose and domain specific modelling languages for processes have evolved. These languages formalize which process elements exist and how they can be composed into a workflow (cf. Meta-Process modelling).

The most well-known and de facto standard graphical notation in the domain of business processes is the Business Process Model and Notation (BPMN 2.0) (http://www.omg.org/spec/BPMN/), which has been under on-going development and extension since 2001. It includes concepts for supporting the process elements mentioned in section 3 and integrates a large variety of additional modelling entities, e.g. conversations and an extensive set of event types. BPMN descends from event-driven process chains (EPC) (Dumas et al., 2005) - another form of process modelling. EPCs are not suitable for modelling complex process structures, though.

The complexity of BPMN has made this workflow language hard to use for non-experts. Due to the large variety of language elements, processes with the same "meaning" can be modelled in a lot of different ways (Wohed et al., 2006). Many of BMPN's elements were introduced in order to fulfil requirements from modelling processes in the business domain, which usually resorts to web services and static calls on the instance level. BPMN does not support the creation of variants of processes or partial processes. As it is our goal to model autonomous workflows for more dynamic and complex heterogeneous environments, we need a more structured and flexible language, which also allows the evolution and extension of models.

The Business Process Execution Language (BPEL) (https://www.oasis-open.org/committees/wsbpel) is similar to BPMN, but it incorporates a more formal way to describe business processes resulting in a stronger execution semantic and therefore better engine support. However, most of the aforementioned drawbacks of BPMN can also be observed with BPEL (Wohed et al., 2006), which therefore does not proof to be suitable for our purpose, as well.

XPDL (http://www.wfmc.org/xpdl.html) is a workflow language intended for interchanging process definitions between different process notations, especially for serializing graphical BPMN models. It is extensible and provides strong execution semantics. However, it is based on BPMN and therefore has similar properties, which makes its application within UbiSys not feasible, too.

There also exists a large variety of further workflow modelling languages developed within an academic or an industrial context. Petri net based languages provide formal execution semantics and verification. Modelling Petri net based workflows is very complex, though, and none of the existing languages provides means for the dynamic allocation of process invocations or the creation of process variants. YAWL (van der Aalst and ter Hofstede, 2005), which is a famous Petri net based language, only supports the modelling of control flow. However, within CPS, we also need to be able to express data flow and assign resources to process steps.

In (Ranganathan and McFaddin, 2004) a workflow execution system based on BPEL is proposed, which is intended to facilitate user interaction with web services in a pervasive environment. This work is mostly concerned with automatic discovery and integration of pervasive web services. It is suitable for solving aspects of the requirements concerning the dynamic integration and deletion of components in a ubiquitous system, but does not consider further requirements in detail.

(Montagut and Molva, 2005) present a workflow management system supporting the distributed execution and dynamic assignment of resources and tasks for pervasive environments. The concept based on BPEL covers several of our requirements, but still does not allow the extension of models and the creation of process variants due to its BPEL-based nature.

When evaluating current workflow languages and management systems with respect to the aforementioned requirements, one finds that the majority of these languages fulfil the requirements listed in section 3 only partially. We therefore propose a new language for modelling workflows within ubiquitous environments.

In designing our own workflow language, we will try to adhere to the basic principles of BPMN and Petri nets. Therefore, we will still be able to support BPMN models and map our process models to higher-level Petri nets. We will also integrate concepts that have been presented within related research to solve some aspects with respect to the requirements presented in section 3.

5. MODELLING COMPLEX AND FLEXIBLE PROCESSES

5.1.1. Process Elements and Information

In order to develop a language for modelling processes within ubiquitous systems, we need to identify the most important elements necessary for formalizing workflows in these environments (Van der Aalst et al., 2003).

- *Process step:* A process step represents an activity or task to be executed.
- *Transition:* A transition represents a unidirectional connection between process steps, creating an ordered workflow.
- *Data:* A data element represents actual data of a specific type being consumed or produced by a process step.
- *Event:* An event represents a certain occurrence of a special happening and can lead to other events or trigger new processes.
- *Logic step:* A logic step is a special type of a process step containing logic for controlling the activation flow of other process steps.
- *Process:* A process contains one or more process steps, transitions, data, events, and logic steps, and can be regarded as the description of a closed sequence of actions.
- *Handling Entities:* A handling entity (resource) is responsible for performing one or more process steps. An entity can be a certain device, a service, and also a human being.

In addition, the process elements mentioned above need to have a certain set of attributes. We will detail this information later on when we describe our concept.

5.2. Meta-Meta-Model for Processes

First of all, we will present the underlying meta-meta-model for our process meta-model. We find that we only need ***Components*** and ***Relations*** as elements for describing the meta-model (Schlegel, 2008).

Components are a well-established concept, e.g. in the field of software engineering, for describing a closed entity providing a defined functionality (Szyperski, 1998). They can be accessed via their interfaces, which describe requirements for using the components and the result of their usage (pre-/postconditions). Components can be composed to larger components and also split into smaller ones up to the point of atomic components. As components provide several positive properties, we will apply this central concept on process steps and processes, which therefore represent instances of components, and use the term “process components” in the meta-meta-model.

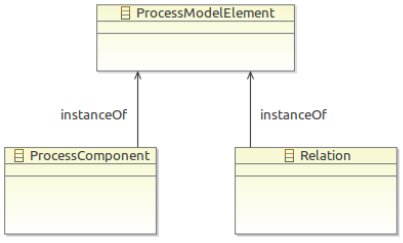

Figure 1: Process Meta-Meta-Model

Relations are used for describing the formal structure of the process meta-model. Using the Unified Modelling Language (UML) as a basis for modelling connections between components, we transfer the object-oriented concepts of inheritance, association, and composition into our meta-model. These meta-model elements therefore represent instances of relations, i.e.

connections between components. Figure 1 depicts this meta-meta-model.

5.3. Meta-Model for Processes

Based on the meta-meta-model presented in the previous section, we can now define the meta-model for processes, which is an instance of the meta-meta-model.

5.3.1. Process Components

The central element of the meta-model is the ***Process Step***, which is the basic component for modelling processes. In our object-oriented model, we differentiate between ***Composite Steps*** containing one or more process steps (depicted by the composition relation *subSteps* in Figure 2), and ***Atomic Steps***. Composite and atomic steps are seen as specializations of a process step (depicted by the inheritance relations in Figure 2). A ***Process*** is regarded as a set of one or more process steps that form a self-contained workflow. This way, processes can themselves contain processes consisting of one or more process steps (cf. composite design pattern) and at the same time, a process can be seen as one step of a super ordinate process (depicted by the *parentStep* association). This modular design leverages extensibility and reusability when modelling complex processes.

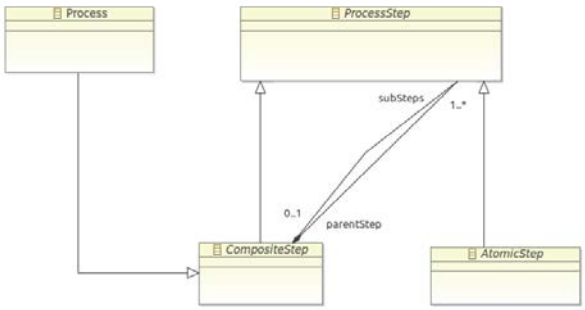

Figure 2: Process Components of the Meta-Model

5.3.2. Component Ports and Flow Relations

In order to describe processes as an ordered control flow and data flow graph of process steps, the meta-model provides transitions between process components. As depicted in Figure 3, we introduce the concept of ***Ports*** as parts of a process step. A port represents an entry or exit point for data or control flow concerning a process step.

At runtime, ports will have an activation state, which will be used to decide on the point of execution of the according process step. The process step will only be executed if all of its start ports are in an activated state.

In general, we differentiate between ***Data Ports*** and ***Control Ports***, which are both specializations of port objects (inheritance relation). Data ports are used for modelling data that are consumed by process steps at their start ports, or that are produced by process steps at their end ports. This concept can be compared to a simple function call within a common programming language specifying in-going data necessary for executing the function, and out-going data as a result of executing the function. Data ports represent data of a certain **Data Type** of a possibly external data type model (*type* association). To support the use of data elements of different types, multiple entry ports (*startDataPorts*) and out-going ports (*endDataPorts*) can be contained within a process step. Data ports will be activated after the successful execution of a process step.

Control ports are used for connecting process steps that do not require a passing of data. Similarly to data ports, diverging control flow can be modelled by using multiple *endControlPorts*. A process step can also contain multiple start ports, which may be connected to multiple preceding process steps. In the end, all data and control ports at the start of a process step need to be activated in order to start the process step execution.

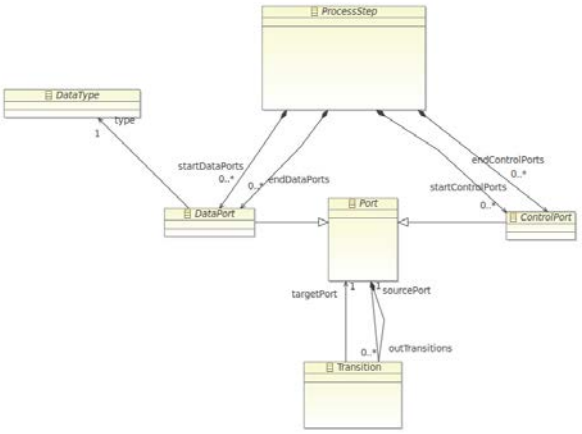

Figure 3: Component Ports and Transition Relation

Connections between process steps are modelled by using ***Transition*** objects, which can be viewed as a relation between exactly one port of a process step (*sourcePort* association) and exactly one port of another process step (*targetPort* association). A transition is defined as part of the port it originates from (composition relation *outTransitions*). This way, a port contains all of its out-going transitions. As the modelling of loops requires additional attention, we will introduce a special process component concept for loops later on. Therefore, transitions are only allowed between the ports of distinct process steps.

The connections between the set of elements of the meta-meta-model and the elements of the meta-model are presented in Figure 4.

When a process step has been executed successfully, all of its end ports become active, which also activates the transitions connected to the respective end ports. In a succeeding step, the transitions' target ports are activated.

When modelling composite process steps, there also needs to be a transition created between the start ports of parent step and its child step, as well as between their respective end ports. Transitions are only allowed between process steps on the same hierarchical

level and their direct parents. This cannot be enforced structurally by the model, but it has to be formalized by using additional constraints, for example by using the Object Constraint Language for object-oriented models (http://www.omg.org/spec/OCL/). Consequentially, we eliminate dependencies between process steps that are not adjacent to each other or in a direct parent-child-relation.

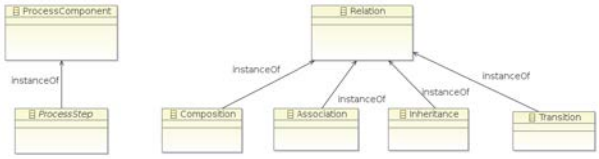

Figure 4: The Meta-Model Elements as Instances of the Meta-Meta-Model

In describing relations between process components in such a manner, we are able to model a flow of process step executions and we can leverage the encapsulation of a closed sub-workflow and its reuse in another process. Figure 5 shows an example of a process model including process steps, which again contain other encapsulated process steps.

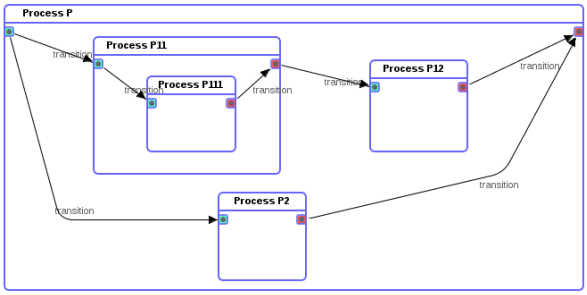

Figure 5: Exemplary Process Model with Nested Processes

5.3.3. Component Specializations

Until now, we mostly described the basic structure of our process meta-model, focussing on process components and process steps respectively, and showing how to compose them. Yet, we need more specific forms of process steps in order to have a comprehensive set of modelling elements. Thanks to the object-oriented approach of modelling the process elements, we can easily extend the previously presented concepts of atomic and composite process steps by inheritance and thereby introduce specializations of process steps. Figure 6 depicts a small set of possible extensions for data and control flow often used within other workflow languages, e.g. BPMN and BPEL.

An extension of the composite step may be used in order to represent ***Loops*** within a process. A loop could again be extended by specialized loop type, e.g. a do-while-loop, containing a loop condition and a loop counter.

Several logic elements for controlling the activation flow within a process are modelled as specializations of an atomic step. In general, a process step will be executed if all of its start ports are in an activated state. This can be seen as the logical AND connective. Other logical connectors for joining the control flow and formalizing a more special activation pattern (e.g. ***OR*** and ***XOR*** connectives) need to be modelled explicitly. In the same way, we can define conditional join operations based on data at the start port of the respective process step (e.g. ***IF***). The forking of an activation flow is modelled by creating multiple transitions from the corresponding out-going port of a process step to the eligible target process steps (see Figure 5).

We also introduced process steps for ***Data Manipulation***. The ***Data Explosion*** component analyses a complex data type and breaks it down into primitive data types. The ***Data Implosion*** step combines primitive data types into a complex type.

For calling external functionality, we added the ***Service Invoke*** component into our model. By further specializing this type of process step, we can support various kinds of service calls, e.g. to REST or SOAP based web services, or to OSGi services, via their respective services addresses or interfaces.

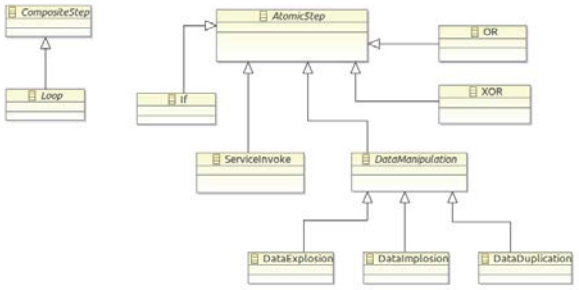

Figure 6: Possible Extensions of Process Steps

Thanks to the object-oriented modelling approach, we can easily extend and further refine the types of process steps via inheritance within the meta-model. In the same way, we can extend the (external) data type model used for defining types of data ports.

5.4. Events and Process Slots

Now we have an extensive set of elements for modelling processes. However, additional means for representing special model elements in ubiquitous processes need to be available.

We introduce ***Events*** as a special type of process step to the model (see Figure 7) to allow for the representation of loosely coupled architectures predominant in cyber-physical systems (Talcott, 2008). Events are viewed and modelled as process steps. The triggering of an event as a consequence of an action within a process is described by creating an event process step and connecting its in-going control port to the control port of the process step responsible for triggering the event. This event can have a special payload and be handled by an event processing engine (cf. complex event processing). The consumption of an event by another process step, as well as the triggering

of other events or processes by the event, can be modelled accordingly. Interaction of and communication between processes is therefore event-based.

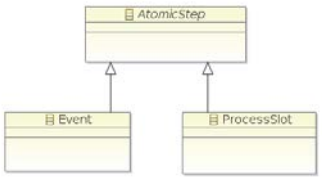

Figure 7: Event and Process Slot Extensions

To allow for the runtime usage of a process step whose implementation is not available at modelling time, we introduce a place holder for process steps called ***Process Slot***. Using this concept, the interface of a process step can be defined without providing a specific functionality. The execution engine is then able to bind the process slot to a specific process step by matching the existing process steps to the slot's ports and name at runtime.

5.5. Creating Domain-specific Processes

Thus far, we defined a domain-independent vocabulary of elements for modelling processes within ubiquitous systems. However, we also need to add semantics to the process components in order to describe the functionality of a process component, and to have more sophisticated means to model and select processes appropriately. Therefore, a *type* attribute for process components is introduced, which represents a domain-specific semantic description of the actions a process actually performs. This could be, for example, a user interaction step or a fetching step in the smart home domain. Backed up by a domain-specific object-oriented model or ontology, we can leverage the properties of this domain-knowledge and create flexible and adaptive process models. Based on the domain model, an execution engine could search from a repository of available process steps for a process step with a matching type attribute.

In using a structured domain-model for the typification of process components, we gain several advantages when choosing an appropriate process step. On the one hand, process step types can be refined via inheritance, e.g. a data input process can be specialized to a speech input, text input, and gesture input process. A fetching process can be specialized to a paper fetching process, as depicted in Figure 8.

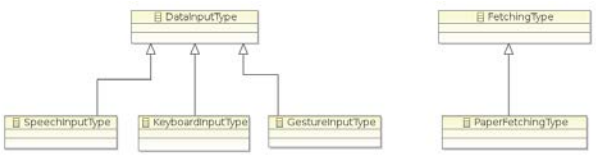

Figure 8: Component Type Refinements via Inheritance

On the other hand, we can make use of polymorphism of process steps, i.e. a process step can be of a certain type and at the same time also of its parent types, which can be continued transitively. At runtime of a process instance, the process engine could walk through the inheritance structures and search for a process step with a matching type or one of its specializations that is suitable and available for execution. For example, a process step providing data input may be required but not specified any further and therefore a speech input process step is used, as it is also of type data input and therefore has the same general properties.

A more sophisticated method of creating and using a domain-specific model would be to use semantic technologies. This would result in more advanced mechanisms for process modelling and selection by using verification and deduction based on logic.

Regardless of which method for modelling the domain-knowledge is applied, creating a comprehensive and structured model for describing the application domain is an important requirement for achieving flexibility in process execution.

5.6. Component Attributes and Roles

In order to meet the requirements described in Section 3 and to complete the meta-model, we introduced a set of further attributes for the process components.

Besides attributes for naming and identifying components on the model and instance level, an optional role-based *handling entity* for a process step can be defined (Montagut and Molva, 2005), (List and Korherr, 2006). This corresponds to the swim lane and pool concepts of BPMN. We can define an entity (resource), again on the instance or type level, that is responsible for executing the respective process step. By using roles of an underlying model for this allocation of process step handler, we are able to orchestrate multiple devices and classes of devices, and also achieve a basic form of access control (Sandhu, 1998). This concept also supports our future goal of distributing process execution across multiple devices in ubiquitous systems.

In addition to the aforementioned attributes considering properties at modelling time, we also need to have component attributes with respect to runtime properties. These include, amongst others, activation states for ports and transitions, as well as an execution state for process steps.

6. MODELLING ENVIRONMENT AND AUTHORING TOOLS

6.1. Technical Realization

The implementation of the introduced process meta-model is based on the Eclipse Modeling Framework (EMF) (http://www.eclipse.org/modeling/emf/), which provides an extensive set of applications and tools for modelling and creating domain-specific languages. This open source framework is based on Java and supports mechanisms for automated source code generation, model verification, and persisting model information with the help of the XMI (XML Metadata Interchange)

format. Thanks to its object-oriented design, we can map our models and concepts directly to objects that can be used for process execution by a corresponding process engine.

6.2. Process Authoring

Apart from implementing the meta-model, we have started developing a toolchain for supporting the computer-based authoring of processes.

6.2.1. Process Editor

Using built-in tools of EMF, table-based editors for Ecore models can be generated automatically. Unfortunately, the complexity and low lucidity of these editors requires the user to have in-depth knowledge of the underlying model. To improve usability in terms of consistency, conciseness, and comprehensibility, we have developed a graphical process editor based on the Graphiti tooling infrastructure for EMF (http://www.eclipse.org/graphiti/).

Figure 9 displays a screenshot of the process editor, which can be divided into three areas. (1) shows the main drawing area for the process model, (2) shows the set of modelling elements available, and (3) shows the components' attributes.

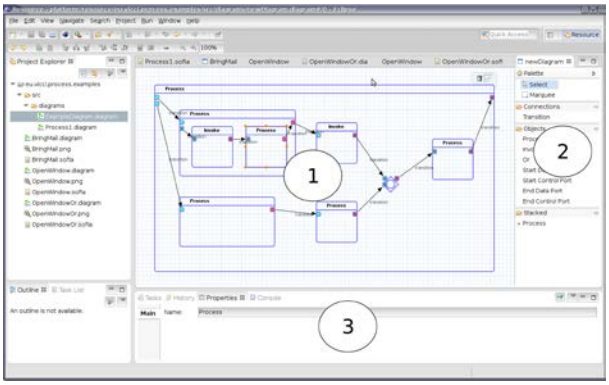

Figure 9: Eclipse-based Process Model Editor

At some points, however, enforcing additional rules for dealing with exceptional combinations of components and relations is necessary during modelling. Formalizing these restrictions inside the meta-model would usually lead to a large increase of its complexity. Constraints that cannot be applied structurally by the model are defined separately using the EMF Validation Framework. After creating a process model using drag and drop from the element list to the main drawing area, a check of the model's validity according to the meta-model and to the separate external constraints is performed.

The result of creating a model is an XML-based representation of the process model including graphical information for visualization of the process model inside the editor and additional process monitoring tools.

6.2.2. Process Repository

In order to model and execute domain-specific processes, we are currently planning on developing a repository for processes and process steps that can be accessed by the editor and the execution engine.

Thanks to the model-based design and modularity, we will be able to use the graphical process editor for the initial creation of processes and process steps which can be submitted to the repository, and to further extend and adapt the processes inside the repository.

At this point, we will also be using a semantic description for processes, their ports, and their domain-specific types to have additional means for checking compatibility of process steps, recommending suitable process steps, and verifying a modelled process. To do so, we are able to draw upon an extensive set of methods from the field of semantic web technologies.

6.3. Process Execution

We have also started implementing a process engine for executing instances of process models based on the meta-model presented before.

The XML-based description of a process generated by the process editor is loaded and validated by the process engine. Afterwards, the engine creates a process instance and walks through the objects defined in the respective process model, calling the methods implemented for handling the specific type of process element.

In order to represent and persist the runtime state of a process, we extended our meta-model with runtime information. Consequently, we also have an extended version of our meta-model for representing the state of process instances (Lehmann et al., 2010).

Process instances are currently executed on a central orchestration server supporting the invocation of web services and OSGi services via remote procedure calls. However, as part of our future work, we will be able to distribute the execution of process steps and complex processes across several servers based on a peer-super-peer network infrastructure (Schlegel, 2009).

7. DISCUSSION

Our aim in designing a process modelling language was to be able to cope with new requirements that come along with the emerging new form of complex systems of systems, called cyber-physical systems.

Despite the complexity of CPS, we tried to adhere to simple structures with respect to the meta-models. Using components as basis for describing process steps, we are able to have modular entities representing one process step, which can be combined into larger, complex process building blocks and reused for modelling. These hierarchical structures help with modelling and visualizing complex processes.

Principles of object-orientation help us with defining connections and relations between process steps on the syntactical level, but also on the semantic level. Based on this structured domain-specific model, the meaning process components, as well as their relations with each other, can be described and used for further semantic processing. However, we have to investigate the feasibility of using semantic

technologies, as processes can contain time-critical components and performance may be an issue with their execution.

Due to the object-oriented approach, we are able to extend our process meta-model very easily without changing the model's core structures. The creation of specializations and variants of processes is also possible via object-orientation (Schlegel, 2009). However, the domain model for describing process components needs to be developed and available before modelling processes.

By using the concept of process slots and defining handling entities, as well as, process steps on the type level at modelling time, we achieve a high level of flexibility when executing process instances. Thanks to the model-based description and to polymorphism, the execution engine can search for suitable process steps and handlers at runtime, walking through the inheritance tree. In case a process handler of a certain type is unavailable or a certain process step cannot be executed, the engine could find a matching replacement within one of its specializations or generalizations. In order to make use of these ad hoc replacement mechanisms, we also need a model for describing process handling entities and to define their capabilities, as well as, the matching requirements for the execution of the respective process steps. A basic form of this validation can already be achieved by checking the process steps interface, i.e. its type, its ports, and its name, with respect to compatibility.

The introduction of events for intercommunication and interaction of processes leverages the integration of loosely-coupled systems and supports flexibility in process execution, as process components can trigger other processes without an explicit representation of this relation in the process model. However, we need additional rules for describing correlations among events and between events and process steps, which have to be handled by an auxiliary event processing engine.

Due to the process component composition, we are able to regard every process component as a self-contained process, which can be executed on a set of distributed process engines. Though, in order to achieve this distribution of process execution, there need to be communication and synchronization mechanisms among the process engines.

We based our design on the core concepts and elements of common workflow languages, e.g. BPMN, and therefore are able to map processes created with similar workflow languages to our model and execution engine. Furthermore, our workflow language facilitates the modelling and execution of more complex and dynamic processes within heterogeneous environments, achieving a high level of autonomy of processes.

However, there are several additional models and rules necessary in order to describe all aspects regarding process types, process execution, and process handlers.

Evaluating our concepts with respect to the requirements presented in section 3, we find that we can meet all the requirements that were introduced as being novel with respect to ubiquitous systems. Related research within an academic and industrial context is currently able to satisfy only a subset of the requirements, as the workflow languages are often too static and lack expressiveness, as well as, flexibility.

In order to evaluate our approach and to show its feasibility, we implemented the process meta-model, as well as, started to implement an execution engine and a graphical editor for process models as first elements of a our toolchain for ubiquitous processes. We will extend and improve our tools and models in the near future.

8. CONCLUSION & FUTURE WORK

In this paper, we presented a new meta-model for formalizing workflows within cyber-physical environments developed from a software engineering perspective. State-of-the-art workflow languages often only support parts of the new requirements introduced by cyber-phyiscal systems. Therefore, we developed a new meta-model for processes, which is mainly based on concepts of object-orientation and of component-based systems. We adhered to the paradigm of model-driven architectures, which yielded several benefits with respect to modularity, reusability, and extensibility of process components. By adding domain-specific descriptions to process components and using semantic models, we achieve a high flexibility when executing processes via a dynamic allocation of process handlers. Thus, workflows become more intelligent and autonomous.

However, there are still several open issues that need to be resolved in order to develop an extensive process toolchain for current and future cyber-physical environments. We believe that with our process meta-model, we laid the foundation for smart autonomous workflows within complex heterogeneous environments.

Our future work will include the development of a process component repository and a semantic domain-model for the classification of process components in the area of smart homes. We will also model capabilities of process execution entities and requirements necessary for executing process steps. In doing so, the process engine will be able to select appropriate process steps at runtime. One step further, we will investigate the usage of agent-based technology in order to find appropriate process steps more intelligently during execution.

We will also investigate the mapping of our process meta-model to concepts used within Petri nets. The advantages of formal verification may prove helpful when constructing and analysing safety critical workflows as they may be required within cyber-physical systems. Hierarchical Petri nets may be suitable for our meta-model to be mapped to.

In order to better adapt workflows to the current situation and environment, we are going to use context information collected by the sensors within the ubiquitous systems and thereby make the processes

context-aware and more intelligent, (Wieland et al., 2008). These adaptations can be directly incorporated into the process models (Yongyun et al., 2007).

The decentralized execution of workflows will also be one of our main focuses with respect to further research activities (Hens et al., 2010). Distributing workflows across several orchestration peers may increase the availability of the workflow system and make the workflows more resilient against failures and outages (Friese et al., 2005).

ACKNOWLEDGMENTS

This research has been partially funded within the VICCI project under the grant number 100098171 by the European Social Fund (ESF) and the German Federal State of Saxony.

REFERENCES

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., Steggles, P., 1999. Towards a Better Understanding of Context and Context-Awareness. *HUC '99 Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pp. 304-307. London: Springer-Verlag UK.

Aßmann, U., Zschaler, S., Wagner, G., 2006. Ontologies, Meta-models, and the Model-Driven Paradigm. *Ontologies for Software Engineering and Software Technology*, pp. 249-273. Heidelberg: Springer-Verlag Berlin.

Scheer, A.-W., Thomas, O., Adam, O., 2005. Process Modeling using Event-Driven Process Chains. In: Dumas, M., van der Aalst, W. M. P., ter Hofstede, A. H. M., eds. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Hoboken, NJ: John Wiley & Sons.

Friese T., Müller, J. P., Freisleben, B., 2005. Self-healing Execution of Business Processes Based on a Peer-to-Peer Service Architecture. In: Beigel, M., Lukowicz, P., eds. *Systems Aspects in Organic and Pervasive Computing, Lecture Notes in Computer Science*. Heidelberg: Springer Berlin Heidelberg, pp. 108-123.

Hens, P., Snoeck, M., De Backer, M., Poels, G., 2010. Transforming standard process models to decentralized autonomous entities. *5th SIKS/BENAIS Conference on Enterprise Information Systems (EIS 2010)*, pp. 97–106. November 16, Eindhoven, The Netherlands.

Lehmann, G., Blumendorf, M., Trollmann, F., Albayrak, S., 2010. Meta-modeling runtime models. *MODELS'10 Proceedings of the 2010 international conference on Models in software engineering*, pp. 209–223. October 2-8, Oslo, Norway.

List, B., Korherr, B., 2006. An evaluation of conceptual business process modelling languages. *SAC '06 Proceedings of the 2006 ACM symposium on Applied computing*, pp. 1532-1539. April 23-27, Dijon, France.

Montagut, F., Molva, R., 2005. Enabling pervasive execution of workflows. *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. December 19-21, San Jose, USA.

Ranganathan, A., McFaddin, S., 2004. Using workflows to coordinate Web services in pervasive computing environments. *Proceedings IEEE International Conference on Web Services, 2004*, pp. 288-295. July 6-9, San Diego, USA.

Sandhu, R. S., 1998. Role-based Access Control. In: Zelkowitz M. V., eds. *Advances in Computers*. Elsevier, 237-286.

Schlegel, T., 2008. *Laufzeit-Modellierung objektorientierter interaktiver Prozesse in der Produktion*. PhD Thesis (in German). Universität Stuttgart.

Schlegel, T., 2009. Object-Oriented Interactive Processes in Decentralized Production Systems. *Human Interface and the Management of Information. Designing Information Environments*, *Lecture Notes in Computer Science* . Vol. 5617: pp. 296–305.

Szyperski, C., 1998. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press.

Talcott, C., 2008. Cyber-Physical Systems and Events. *Software-Intensive Systems and New Computing Paradigms*, pp. 101-115. Heidelberg: Springer-Verlag Berlin.

Van der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B., Barros, A. P., 2003. Workflow Patterns. *Distributed and Parallel Databases* Vol. 14: pp. 5-51.

Van der Aalst, W. M. P., Ter Hofstede, A. H. M., 2005. YAWL: yet another workflow language. *Information Systems* Vol. 30: pp. 245-275.

Weiser, M., 1991. The computer for the 21st century. *Scientific American*, Vol. 265: pp. 94-104.

Wieland, M., Kaczmarczyk, P., Nicklas, D., 2008. Context Integration for Smart Workflows. *Percom 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pp. 239-242. March 17-21, Hong Kong.

Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., Russel, N. 2006. On the Suitability of BPMN for Business Process Modelling. *Lecture Notes in Computer Science* Vol. 4102/2006: pp. 161-176.

Yongyun, C., Kyoungho, S., Jongsun, C., Jaeyoung, C., 2007. A Context-Adaptive Workflow Language for Ubiquitous Computing Environments. Computational Science and Its Applications – ICCSA 2007, *Lecture Notes in Computer Science* Vol. 4706: pp. 829-838.