ABSTRACT
We have found that different process algebraic composition techniques, combined with consideration of the restrictions on the ability of different parts of a system to share data and state, provide a basis for identifying abstractions at the Platform Independent level of modeling. The paper presents our ideas and is aimed to initiate a discussion about the basis for identification of abstractions and the related areas of composition, reasoning and interface specifications, at the platform independent level.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques; D.2.1 [Requirements, Specifications]; ]; D.3.1 [Formal Definitions and Theory]; 

Keywords
Platform Independent Modelling, abstractions, CSP composition, CCS composition, reasoning

1. INTRODUCTION
Dividing a system model into modules representing abstractions is an accepted principle to conquer complexity at any level of modelling. Such abstractions as aspects [15], components and services [3, 9] were born as platform specific abstractions. Later it was recognized that identification of aspects at the requirements and architecture level, i.e. at the Platform Independent level [13], may give advantages in system evolution [10], in particular if the separated aspects ensure the temporal properties of the models with which they are composed. We believe that the same conclusion can be applied to separation and composition of other types of abstraction.

Although the advantages of early separation of abstractions are widely understood, the results achieved in this area are related to the Platform Specific level of modelling. Different categories of aspects are defined using state graphs reflecting the AspectJ constructions [7, 17]. Presentation of other abstractions also suggests some implementation knowledge, such as the presence of methods or functions that the software system performs [5, 9]. Modular reasoning [6] in aspect-oriented software development is also defined for AspectJ. When it comes to discussion about reasoning in aspect-oriented programs, there is always a question about the interfaces and implementation dependencies that developers should specify [7, 11]. As the interfaces and dependencies may be seen both at the Platform Independent and Platform Specific levels of abstraction, they may relate to both levels of modelling.

At the Platform Independent level the models should not be "about functions that the software system must perform. The models should be about abstractions built in the context of business operations" [12]. As modelling methodologies do not give guidance on the criteria for abstraction separation at the Platform Independent level, in this paper we propose our criteria and pose a question: Are there other suggestions for separation of platform independent abstractions? The aim of this paper is to initiate a discussion within the modelling community about abstractions, composition and reasoning.

In order to present our ideas in a familiar framework we define a UML profile for Platform Independent Modelling and extend the UML semantics with the notions that are needed to separate abstractions. Among those extensions are:
- the CSP (Communicating Sequential Processes) [8] parallel composition technique extended for models with data and with the rules for dealing with specification violation;
- the CCS (Calculus of Communicating Systems) [16] composition technique applied for models with data;
- the notion of state derivation, related to the UML notion of attribute derivation, but more general;
- the notion of an event being a unit of an elementary type, which is different from the current interpretation of events in the UML as classes [14].

The structure of the paper is the following. Section 2 presents a case study we use to illustrate the ideas of the paper. Section 3 defines a UML profile for Platform Inde-
pendent Modelling used for separation different abstractions based on the process algebraic composition techniques. Section 4 draws conclusions and poses some questions.

2. CASE STUDY

Let us try to identify different abstractions in a case study. It is a Public Information System that keeps up-to-date the information about current education types and the possible exemptions from courses of one education for those who completed another education. As the information is supplied by different authorized official organizations, each entry goes through a security check. Moreover, the information about new educations and available exemptions must be published on an official publisher-site. The publisher-site deals with information coming from different sources, one of which is our information system. The publisher-site has its own restrictions on publishing and priorities defined by legal regulations and other business rules.

Modelling this system we can identify Exemption as a key abstraction. However, an exemption is defined in terms of a pair of educations, so the abstraction Education is also required. We also recognize a Security Check abstraction. The publishing of the information may be represented using an abstraction Publisher.

Analyzing the relations between abstractions, we notice that the Education abstraction recognizes the following events:

Register-Education Modify-Education
- Date : Date; - Date : Date;
- Name : String; - Name : String;
- Type: String; - Type: String;
- Contact: String; - Contact: String;
- Courses: Strings; - Courses: Strings;

Discontinue-Education
- Date : Date;
- Name : String;
- Type: String;
- Contact: String;
- Courses: Strings;

- The Exemption abstraction needs two instances of the Education abstraction. The events recognized by the Exemption abstraction are also shared with the Education abstraction:

Set Up Exemption Modify Exemption
- Name: String; - Name: String;
- ThisEducation: Education; - ThisEducation: Education;
- AcceptedEducation: Education; - AcceptedEducation: Education;
- Exemption Courses: String; - Exemption Courses: String;

Remove Exemption
- Name: String;

The Security Check abstraction presents behaviour that required before any behavioral step of instances of Education and/or Exemption to prevent unauthorized modification. The events recognized by the Security Check are:

Secure Event = All events of classes Education or Exemption;
Set Password Enter Password Reset
- Saved Password: String; - Pass: String;
- Pass: String;

We may make the Education and Exemption responsible for sending the information about state changes to the Publisher. However, a better design is to introduce a Notifier abstraction that monitors any change of Educations and Exemptions and sends messages about these changes to the Publisher.

The Education and Exemption abstractions can be categorized as object types, the Security Check can be categorized as an aspect. The Publisher abstraction and the composition of {Education, Exemption, Security Check, Notifier} abstractions are categorized as services. In the next section we present the criteria for identification of abstractions.

3. A UML PROFILE FOR PLATFORM INDEPENDENT MODELLING

In order to show our separation criteria in a familiar framework, we define a UML profile for Platform Independent Modelling. A specification in our profile is a triple of an event view, a static view and a dynamic view.

Event View. An event is a happening produced in the environment or generated by an abstraction. An Event View presents event types as tuples Event = (Name, Attributes). Event types are unique and modeled using UML datatypes.

Static View. A static view is similar to a UML class diagram but has a different semantics. An abstraction in this view is a tuple

Abstraction = (Name, A, S, E),
- \( A \) is a finite set of attributes. The set can be empty.
- \( S \) is a finite, not-empty set of states.
- A state may be stored or derived.
- A stored state is a set of values of all attributes. A derived state [2] is calculated using the attribute values of this abstraction and other abstractions.
- If the attributes of abstraction \( B \) are used to calculate the state of abstraction \( A \) then \( A \) “can read state” of \( B \), however cannot change the state.
- \( E \) is a finite, not-empty set of events of the event types from the Event View.
- An event may be an environmental (+a) or a communication action. A communication actions is either a send action a or a receive action a.

If an abstraction that is the source of event a is out of the scope of modelling, then a is treated as an environmental action a.
- If an abstraction that is the source of event a is in the scope of modelling, then it is required that the source has the send action a. Other abstractions may have the complementary receive action a.

An abstraction is depicted as a box (Figure 1) and may be of three stereotypes: (Object Abstraction), (Aspect Abstraction) and (Service abstraction). Depending on requirements and the creativity of the designer, a system model maybe built using any combination of objects, aspects and services.

- An Object Abstraction does not distinguish between environmental and communication actions. Send actions and receive actions are treated as different actions, so that a and a are different events.
Figure 1: Static and Dynamic views presenting different abstractions
Objects can Read State of other objects but cannot change their state. For example, object Education reads state of Exemption. We depict this relation using a dashed arrow (Figure 1).

- An Aspect Abstraction is an Object Abstraction that has generalized events and generalized states.
  - A generalized event is an alias that presents each of events from a set. Each of the events of this set causes the same behaviour of the aspect.
  - A generalized state is an alias that presents each of states from a set of states of this and other abstractions. Each of those states has the same input and output transitions of the aspect behaviour.

The Security Check is an example of an aspect abstraction with an abstract event Secure Event. Each of events of Education and Exemption is a Secure Event specified in the Security Check aspect.

If the intersection of the sets of events of a set of abstractions is not empty, then these abstractions are composed using the CSP composition rules extended in [2] for models with data.

The extended CSP composition rules may be considered as an abstraction itself; the abstraction that recognizes a generalized Event representing any event of composed abstractions and can read state of all composed abstractions.

However, the CSP composition is the same for any system and should not be specified. We will describe the extended CSP composition rules extended in [2] for models with data. The extended CSP composition rules may be applied only to communication actions [16].

- !Send Event representing any possible send action and
- !Receive Event representing any complementary receive action, and
- defines the transitions of the sender and the receiver abstractions after the composition.

However, this abstraction is the same for any system and should not be specified. We will describe the CCS composition rules in the subsection about the dynamic view. At the static view the CCS composition label is shown in a box with round corners. Two composed services are attached to this label by two parallel lines with arrows (Figure 1).

All abstractions are composed using two composition techniques: the CSP composition rules and the CCS composition rules.

- The CSP composition rules are applied to all events.
- If abstractions have complementary communication actions !a, ?a, these actions are considered by the CSP composition rules as different: x !a, y ?a.

The CCS composition rules may be applied only to the complementary communication actions, so the “renaming wrappers” x and y are removed from the actions in order to reveal complementary communication actions !a, ?a. After this operation some of objects, aspects or the results of their composition can be seen as services and composed using the CCS composition rules.

Dynamic View.

At the dynamic view an abstraction (an object, aspect or service) is a non-empty finite set of protocol machines

\[
A = \{PM_1, PM_2, ..., PM_p\}, p \in N
\]

For example, abstraction Education contains one protocol machine. Abstraction Security Check has two protocol machines corresponding to it: Password Management and Control and Password Handler.

A Protocol Machine is a tuple \( PM = (S, D, E, T) \),

- \( S = \{s_1, s_2, ..., s_n\}, n \in N \), is a non-empty finite set of stored states. A stored state has a corresponding set of attribute values, including the state name.
- \( D = \{d_1, d_2, ..., d_k\}, k \in N \), is a finite set of derived states calculated using the states of the machine itself and other protocol machines. \( D \) can be empty.
- \( E = \{e_1, e_2, ..., e_m\}, m \in N \) is an alphabet of events, i.e. a non-empty finite set of recognized events: environmental events and communication-events.
- \( T = \{t_1, ..., t_p\}, p \in N \) is a set of transitions. A transition t can be of types \( t = (s_1, e, s_2) \), \( t = (d_1, e, \text{any state}) \) or \( t = (\text{any state}, e, d_2) \).

A protocol machine is presented as a set of protocol state diagrams, where

- a stored state is depicted as a one-line ellipse. For example, state Active is a stored state of machine Password Management and Control;
- the values of attributes in a stored state are presented near the state in a bubble, as it is shown for Password Management and Control;
- a derived state is presented as a double line ellipse. For example, states \textit{Idle}, \textit{Passed} and \textit{Failed} are the derived states of machine \textit{Password Handler};
- the rules of derivation of a derived state the generalized events and states are presented as expressions of the protocol machine; The examples of the expressions can be seen in the specification of \textit{Password Handler};
- a transition is depicted
  - as an arc, i.e. a pair of states labeled by the events that cause this transition or
  - as an arrow coming from or to a derived state and labeled by the events that cause this transition.

A Protocol Machine behaves as follows.

1. If a Protocol Machine cannot undergo any change of state, it is quiescent.

2. 

    \textbf{CSP }\parallel\textbf{ composition rules for machines with data.} 

    - When a Protocol Machine is presented with an event it can reach a new quiescent state.
    - When presented with an event that is not in its alphabet, a Protocol Machines ignores it.
    - When presented with an event \(e \in E\) that is in its alphabet, a Protocol Machine either accepts it or refuses it. Acceptance or refusal of an environmental event by machine \(A\) is determined by:
      - the state of \(A\) before the event and after the event; and
      - the state of other machines state of which \(A\) can read but cannot change.

For example, Protocol Machine \textit{Education} (Figure 1) being in state \textit{Registered} refuses events of type \textit{Register}, but accepts events of types \textit{Modify Education} and \textit{Discontinue Education}. 

- If all machines that have \(e\) in their alphabet accept \(e\), this event is accepted.
- If at least one of those machines refuses \(e\), \(e\) is refused.

We want to emphasis that there is a novelty in the way Protocol Modelling uses CSP parallel composition. CSP parallel composition defined in the context of process algebra uses monadic symbols and the processes are algebraic (no data).

As in the algebraic context, CSP parallel composition in PM preserves determinism of the underlying machines, so the behaviour of a protocol machine can be specified as the set of traces of accepted events.

For example, event \textit{SetExemption [This-Education, Accepts-Education]} will be accepted if
- two instances of protocol machine \textit{Education}
- \textit{Education:This-Education} and \textit{Education:Accepts-Education} are in state \textit{Registered};
- the instance of protocol machine \textit{Exemption [This-Education, Accepts-Education]} is in state \textit{SetUp};
- the \textit{Security Check} is in state \textit{Active} and \textit{Passed}.

3. 

    \textbf{CCS composition rules for machines with data.} 

    (a) - If protocol machines \(A\) and \(B\) distinguish communication actions (they may be called services);
      - \(A\) is in the state where it can produce send action \(!a\);
      - \(B\) is in the state where it can accept the complementary receive-action \(?a\);
      - then action \(a\) can take place;
      - \(A\) goes to the output state of the transition labeled by \(!a\);
      - \(B\) transits to the output state of the transition labeled by \(?a\).

    (b) If \(A\) is in the state where it cannot produce the send action \(!a\), no transition happens.

    (c) If \(A\) is in the state where it can produce the send action \(!a\) and there is no reader \(B\) able to engage in \(?a\), then the writer \(A\) is “blocked”.

There is a novelty in the way Protocol Modelling uses CCS composition. The CCS composition is used on events that have data and between processes that have state and data, whereas in CCS process algebra the events are monadic symbols and the processes are algebraic (no data).

CCS composition is non-deterministic. If two communications can happen, one is chosen non-deterministically. The other communications that were possible may still be possible, depending on the effect of the one that happened.

Services \textit{Information System} and \textit{Publisher} are composed using the CCS composition rules.

4. \textbf{ABSTRACTIONS AND REASONING CONCLUDING QUESTIONS}

In this paper we have proposed two criteria for separating abstractions at the PIM level of modelling:
- the abstractions on events that cause application of different composition techniques and
- the ability to read state of other abstractions.

Also we have proposed a PIM UML profile with two different composition techniques. The interfaces of abstractions in this profile are defined in terms of events. The dependencies are defined in terms of states. The models in this profile are executable models. For example, the ModelScope tool [1] generates Java code from the meta-description of the PIM protocol models. The models can be implemented in any other programming language as well and can be used for simulation.

The most useful outcomes of the proposed separation of abstractions are
- the knowledge about the system behaviour determinism or non-determinism already at the Platform Independent level;
- the knowledge about the borders of local reasoning on models.

The \textit{CSP }\parallel\textit{ composition technique applicable for composition of objects and aspects results in deterministic behaviour [2] being a set of sequences of events. As it has been proven in [4], the result of the composition possesses the property of observational consistency or local reasoning. \textit{Local reasoning} is the ability of understanding some properties of system behaviour based on examining its abstrac-
tions one by one. Local reasoning relates properties of an abstraction to properties of the result of the composition. For example, only examining the behaviour of the Security Check aspect, it is possible to say that the sequence ?Set Password; ?Enter Password; ?Enter Password; ?Modify Exemption does not belong to the behaviour of the modeled system. It is not possible to enter password twice without a reset.

Local reasoning even allows reasoning when there is no formal statement of properties or requirements. In this case the behaviors of each of abstractions are used as properties that the behaviour of the result of composition of those abstractions should have.

The CCS composition results in non-deterministic behaviour because if two communications can happen, one is chosen non-deterministically. The other communications that were possible may still be possible, depending on the effect of the one that happened. It is impossible to reason locally on models composed using the CCS composition. The CCS composed model can be understood only by analyzing the result of the composition as a whole.

The criteria for separating abstractions at the PIM level that we have proposed may be other possible criteria and the consequences of their choice need a discussion that we would like to initiate by this paper.

5. REFERENCES