Integrating Joint Behaviour and Dialogue Description

Gavin Doherty and Michael D. Harrison

Human Computer Interaction Group,
University of York
Email:{gavin,mdh}@cs.york.ac.uk

Abstract. Agents are becoming increasingly common as a means of structuring interactive systems, due to the highly complex and concurrent nature of modern systems. The manner in which interaction between these agents is specified is of fundamental importance, and must pay heed to expressivity and reuse concerns. There are also concerns specific to interactive systems, and in particular the need to specify and reason about user-system dialogue. We have shown previously that the standard model of object interaction is inadequate with respect to these concerns, and that the action model performs better with respect to these criteria. In this paper these results are drawn together with approaches previously taken in interactive systems. From this basis a schema calculus with interleaving semantics is proposed, which better addresses the concerns of expressivity and reuse in the interactive systems context.

1 Introduction

Modern interactive systems are large, complex entities comprising a number of components with the capability of handling multiple simultaneous threads of dialogue. In this environment, the concept of *object* and the concept of *agent*, an independent active object, has become crucial, as it affords us an ability to manage this complexity.

By specifying the system as a collection of co-operating agents operating concurrently, further expressive power has been gained. Defining the manner in which the agents co-operate and communicate is of fundamental significance in harnessing this power. When choosing a specific model for concurrency and co-operation, there are two issues of particular importance, namely those of *expressive power* and of *reuse*.

Furthermore, there is a need to consider the goodness of fit in relation to the concerns of the domain of the system under consideration. Our own interest is in *interactive systems* and hence issues of *behaviour* are of importance. For example, the order in which operations can be invoked (the user input syntax) is of particular relevance, as this must relate to the user's tasks and goals, and its structure affects issues such as the likelihood and consequences of user errors.

We have previously illustrated the failings of the standard model for object interaction, the event model, with regard to reuse and expressive power [5]. We have explored an alternative to the traditional event model, based on the concept of *actions*, in a general software engineering context, and shown how it copes with the reuse aspect.

The next challenge is to apply this knowledge in an interactive systems context in a practical fashion, keeping in mind our commitment to specification reuse and to tractable usability reasoning. In the following sections we shall consider in turn the concerns of interactive systems, reuse, and the concept of enhancement, and show how these concerns may be addressed by means of a new schema calculus.

1.1 Interactive System Concerns

In the domain of Interactive Systems, we are naturally concerned with the interaction between the user and system. In this context state, which is visible in the presentation or which affects the actions which may be taken by the user, is of particular importance.

The difficulty which arises is the interdependence between this **dialogue description**, and the **behaviour**, and functional dependencies of the system. If the two are separated then **consistency problems** arise, also the **separation of concerns** between the components of such a specification can be problematic. Yet if we seek to use a single notation, then it must be one expressive enough to encompass both dialogue and behaviour. We consider three previous hybrid approaches [1,14,2] in the domain of interactive systems which attempt to cover both dialogue and function description. These provide a concrete basis for the discussion and exemplify interactive systems concerns.

Behaviour and Dialogue The behaviour of an interactive system generally concerns when and under what conditions certain events can happen. Those events which directly involve the user (user input and system output, ie. the *dialogue*) are of most concern. The means by which this dialogue is specified determines the ease with which usability reasoning can be carried out.

In a scheme designed to allow designers to *"specify formally the structure of complex and concurrent dialogues"*, Alexander [2] uses CSP [13] behavioural definitions to describe user input syntax, linking operations to each event. CSP provides a rich set of operators for combining events and processes, the main operators being prefix, choice, sequence and parallel composition. User inputs and system responses are represented as atomic events, but causation of events and the communication between components are unspecified. The system model consists of an association between events and a specification in *me too* (a lisp-like executable specification language based on VDM [15]). The events themselves are specified with eventISL (interaction specification language for events). An event describes only one interaction with the user, an interaction consisting of one input from the user and one output from the system. Thus no real model of the system is provided apart from the fact that functionality is structured via events with a single input and single output.

Alexander's CSP approach covers only the dialogue structure aspect of the interactive system specification, and is not sufficient for the task in hand in unaugmented form. Nevertheless, the interleaving semantics of CSP provides a powerful means for specifying and reasoning about concurrent operations without the complexity overhead of computation based models of parallelism.

Hybrid Specification As CSP does not provide support for describing either the structuring of the system as a collection of objects, nor for defining the behaviour of these objects, an obvious step is to combine it with a mechanism for defining an object's state and behaviour. There are a number of difficulties with the use of hybrid notations:

– the problem of maintaining consistency between the two components of each specification,
– no overall meaning can be given to a specification unless there is a semantic link between the two components,
– there can be ambiguity in the *separation of concerns* between the two components (explored below),

– using two notations adds to the complexity of construction, analysis and refinement, and also makes automated support for these more difficult to realise.

This criticism of hybrid notations is not to be confused with the construction of multiple (partial) models of a system as advocated in [9]. As the specifications under discussion define the joint behaviour of a set of interactors, it is desirable that a single specification be constructed for this purpose.

The agent language of Abowd [1] is illustrative of a move towards object based specification, in the form of independent agents. Abowd pays particular attention to the composition of agents to form complete systems, and provides a means of specifying interleaving of agents. Agents include internal and external components, the internal component detailing the operations and state changes of the object, and the external specification (a CSP behavioural specification) detailing interactions with other agents. Although there is a semantic link between the components of the specification, the notation is a hybrid with the attendant problems thereof. Also, as pointed out by Hussey [14], the composition operations deal only with entire agents rather than with individual operations, severely limiting the opportunities for reuse.

Hussey [14] harnesses the expressiveness of the formalism for dialogue description within a more complete specification process through the application of interactor style specifications: using CSP for user input syntax and Object-Z[8] for state and operations. The events (alphabet) in the CSP dialogue description each correspond to a single Z schema. Additionally, each interactor[6] has a behavioural description (alphabet and trace definition) associated with it. In addition to the lack of a semantic link between the components, the lack of integration of the external specification into the object model means that the standard object mechanisms for reuse cannot be applied to external behaviour. There is also the problem of consistency. Whereas the CSP specification gives the sequence of allowable user actions, the object definition could disallow all such sequences completely through the precondition on the operation schemas.

Separation of Concerns With hybrid notations, a difficulty is encountered in obtaining a clear separation of concerns between the two components of the specification. For example, it is possible to express the same behaviour using different combinations of CSP and schema definitions. Consider a simple system which can be either locked or unlocked, with an initial state of unlocked, and two corresponding operations:

$lockstate = \{locked, unlocked\}$

$$\begin{array}{|l}\hline \multicolumn{1}{l}{lock_1} \\ s : lockstate \\ \hline s' = locked \\ \hline \end{array} \qquad \begin{array}{|l}\hline \multicolumn{1}{l}{unlock_1} \\ s : lockstate \\ \hline s' = unlocked \\ \hline \end{array}$$

$SYS_1 \mathrel{\widehat=} lock_1 \rightarrow unlock_1 \rightarrow SYS_1$

$$\begin{array}{|l}\hline \multicolumn{1}{l}{lock_2} \\ s : lockstate \\ \hline s = unlocked \\ s' = locked \\ \hline \end{array} \qquad \begin{array}{|l}\hline \multicolumn{1}{l}{unlock_2} \\ s : lockstate \\ \hline s = locked \\ s' = unlocked \\ \hline \end{array}$$

$SYS_2 \widehat{=} (lock_2 \rightarrow SYS_2)[](unlock \rightarrow SYS_2)$

Clearly the same set of possible traces of behaviour is described by the two definitions, but whereas the first uses the external CSP expression to ensure legal behaviour, the second uses a combination of CSP and precondition schemas.

Summary While all three approaches use CSP as the basis for describing joint behaviour, there are significant differences in the mode of application. Abowd includes communication and a higher level notation of agent interleaving, while Hussey relies on a one-to-one mapping between user input events and operation schemas. Object-Z provides extensive object support, whereas Abowd is limited to structuring only, and Alexander provides none at all. A table summarising the three approaches is given below: Hussey's work effectively illustrates that the expressive power of this combination

Approach	Alexander	Hussey	Abowd
Joint Behaviour	CSP	CSP	CSP+Agent language
System Behaviour	meToo (VDM based)	Object-Z	Agent language
Object Support	none	good	weak
Semantic Link	no	no	yes

Table 1. Comparison of approaches

is very good and that the size of specifications, including both these components, is not unwieldy. A notation which afforded us this degree of expressive power within a consistent semantic framework, would contribute significantly to the development of a pragmatic approach.

1.2 Reuse Concerns

Elaborating on the reuse aspect, we are concerned with reuse in the presence of inter-object (multi-agent) behaviour. Specifically, a serious reuse problem emerges when inter-object behaviour is specified in an event style. As we are concerned with systems comprised of multiple objects, we are interested in the ways in which better performance might be achieved with respect to reuse, as is the case with action based systems.

Reuse Criteria To help structure our analysis of the reuse problem, three views on the reuse problem are utilised, labelled:

– **contextualisation** as observed by Mili [17], in the object model all behaviour must be attached to an object or class, and hence all interaction between two objects must be specified as an operation on one of the two, thus decreasing the reusability of that object.
– **viscosity** pertains to the fact that although changes to inter-object behaviour are very likely to be needed so that the object can be reused, such changes are difficult to carry out in the standard model.
– **dependency** concerns the degree of unnecessary dependency introduced into an interaction by the model of inter-object behaviour, for example the need to specify explicit interaction trajectories in components whose behaviour does not depend on them.

Evaluating the approaches in section 1.1 above with respect to these criteria, we see that the agent language fares badly, concentrating as it does on entire agents as the main unit of specification, and the lack of provision for *alterations* of inter-object behaviour. Hussey's approach performs better as a large class of changes can be encompassed by rewriting the CSP syntax definition, but there is little scope for reuse of inter-object behaviour.

Action Models In contrast to the event model, action based systems suffer relatively little from the above problems. In the action model, a specification consists of a collection of objects and actions. A number of objects can participate in an action, and actions may place preconditions on these participants. All state changes are effected by means of actions. A comprehensive survey in [16] shows the action model to have a high degree of expressive power according to a broad range of criteria. Summarising the work in [5], there a number of features of systems such as DisCo [11] (an object oriented action based language) which contribute towards resolution of the above problems.

- **contextualisation:** the separation of object definition and interaction in a semantically consistent way through the use of actions moves us away from a purely object based system. Hence the context dependence of objects including inter-object behaviour is addressed, at the price of a very different specification architecture.
- **viscosity:** the ability to alter inter-object behaviour through strengthening of guards and extension of states. Although again not unique to action based systems, the use of action style specification makes this type of change more likely to encompass a larger proportion of desired changes to inter-object behaviour.
- **dependency:** the environmental selection of actions based on the availability of appropriate objects satisfying the preconditions. This is the basis of the action style, and allows specifications to be written in such a way that we need not define the sequence of operations which put the system into a certain state in order to define an operation which is enabled in that state.

The key concepts then, are the environmental selection of actions, the ability to strengthen and alter action guards, and the shift in focus away from the objects alone.

1.3 Enhancement

The concept of enhancement is of particular interest. For our purposes we can define enhancement as adding *observable behaviour* to a system specification in a manner which preserves properties of the original system. Following Bramwell [3] we agree that the designer often works by *enhancing* a previous specification, and with this in mind, we will take enhancement as our mode of reuse. Such an incremental mode of development is of particular importance in the interactive systems context, given the modern emphasis on rapid prototyping and fast feedback into the design process over many iterations of the development cycle.

Past approaches to enhancement have generally focussed on data refinement and state [3]. While there is a relationship between refinement and enhancement, as noted by Bramwell, refinement can include simplification or elimination of observables and hence not all refinements are enhancements. We are also interested in *behavioural* enhancements which concern the *trace* of a composite operation. The notion of trace allows us to formulate definitions of enhancement which are meaningful in an interactive systems context. This focus can also be used to resolve the distinction between *composition refinement* and *enhancement*. Bramwells' definition, based on He [12], requires that the traces of the refined interactor yield a subset of those of the original interactor when projected onto the alphabet of the original (a formal definition is given in section 2.4). This can involve the loss of observable behaviour, so we introduce the notion of *composition enhancement* in which the traces of the enhanced interactor yield *exactly* those of the original when projected onto the alphabet of the original.

A challenge posed by this is to revisit such formalisations of enhancement in the context of object oriented specification, for which we need a framework in which inter-object behaviour is defined in such a way that we can reason about traces of behaviour.

1.4 Other Concerns

Action systems have the ability to express multiple object participation in composite operations. This is a powerful model for object interaction, and satisfies the concerns in the object-oriented literature regarding the **expressive power** of the event model. We do not advocate direct use of action models for a number of reasons; on a practical level, use of such models for specification is not widespread at the present time. Also, within the field of interactive systems specification, the impact of the use of such formalisms on usability reasoning is not known.

Model based specification languages such as Z[18] and VDM[15] are a more practical alternative as they are widely used and supported. Although they are not based on the event model, event style specifications using them have been common [7]. As they model the state and operations of the system we can usefully reason about modes, goal achievement, functional invariants and information to be presented to the user. We are particularly interested in object based variants of these mainstream specification languages for the reasons previously discussed (namely, object reuse and ease of implementation). A pragmatic approach would be to incorporate action concepts into a framework based on a conventional specification language, taking into account the concerns of interactive systems.

Above, the examination of interactive system concerns has suggested that an expressiveness similar to CSP behavioural definitions is desirable (particularly for concurrent dialogues), but must be augmented to describe joint behaviour. Previous approaches to this have involved hybrid specifications, which cause consistency and tractability problems. Likewise the reuse concerns mitigate against purely event based specification of joint behaviour, favouring more action based approaches. Our goal is to factor both of these sets of concerns into any proposed approach.

2 A New Schema Calculus

Motivated by the analysis above, we take Object Z [8] and the schema calculus as the basis for our approach, and demonstrate that this approach satisfies the above concerns. Examination of these concerns also allows us to justify the differences between our approach and previous approaches such as that of Hussey [14]. Object Z is a natural choice as it incorporates many object oriented concepts and is based on a widely used model-based formalism (Z). Operations may be declared in much the same way as schemas in Z or as composites of operations. A class encapsulates both state and operations.

2.1 Defining joint behaviour with the schema calculus

As Hussey notes [14], Object Z classes provide a convenient means for structuring a system as a collection of interactors. The attributes of the class correspond to the state of the interactor, with state transitions effected by means of operations. Rather than relying on an external description of the allowable traces of such operations, as was the case in the approaches examined earlier, we define these traces using the schema calculus.

Taking the basic schema calculus operators of choice ([]) and sequential composition ($\fatsemi$), inter-object behaviour and syntax can be defined simultaneously, as the schema calculus expression determines both the flow of data and sequencing of operations. For example the CSP definition:

MENU = (display $\rightarrow$ selection $\rightarrow$ (MENU [] SKIP))

will have an equivalent effect to the schema calculus expression:

$menu \mathrel{\widehat=} display \fatsemi selection \fatsemi (menu [] skip)$

2.2 Separation of Concerns

One aspect of the action model which must be considered carefully concerns the need for moding state to sequence internal operations. This state is neither visible nor meaningful to the user. However an opportunity is present to resolve both this problem and the difficulty in obtaining a clean separation of concerns identified above.

Essentially, the problem of separation of concerns is that moding can be expressed either by means of schema preconditions (guards) or the schema calculus. Given that user visible moding is of particular interest, we formulate the rule that user-visible moding be expressed through the use of guards, and that where possible ‘internal’ modes be expressed by means of the schema calculus. This allows us to achieve a more distinct separation of concerns and should also yield more structured specifications.

For example, a paste command in an editor might be specified as the following:

$$paste \mathrel{\widehat=} File.isWritable \fatsemi Clipboard.getContents \fatsemi File.Insert$$

Where $File.isWritable$ is a precondition on the mode of the file (essentially a guard on the command), $Clipboard.getContents$ returns the contents of the clipboard and $File.Insert$ inserts the text returned by the clipboard.

2.3 Defining Behaviour

Under the semantics of Object Z, sequentially composed schemas are equivalent to atomic schemas. The existing ‘parallel’ operator is actually a form of conjunction with communication, with no concept of visible intermediate states or interleaving.

This is entirely suitable when describing functionality. In the interactive systems context, however, the specific *behaviour* is of interest, and the lack of parallel operations is a problem. As all operations are effectively atomic, we cannot meaningfully discuss intermediate states or simultaneous operations. At best, the possibility of simultaneous operations can be encoded by introducing state variables for each operation, storing the current point in the operation. The operations are then split into sub-operations with preconditions over this state to sequence each sub-operation. This is obviously not an ideal solution. Firstly the abstraction for operations no longer matches that provided by the notation, secondly we have had to encode the execution model explicitly, and thirdly the granularity has been forced to the lowest level.

What is needed is an expressiveness akin to the interleaving parallel semantics of CSP. With such an approach, the schema calculus could be used to define composite behaviour in a way which allows us to define and reason about intermediate states and concurrent operations, without having to resort to hybrid specifications.

In order to achieve an interleaving semantics, there must be a notion of atomicity of sequentially composed schemas. Thus the meaning of a given schema expression can be given in terms of the meaning of a set of sequentially composed schemas, with simple operations defined as sequences of length one. With this we can have a notion of intermediate states within a schema expression, that is, the state at a point in time determines the simple or 'atomic' schemas (within a sequential composition) which may occur next. In our semantics, we allow either an incomplete operation to be continued, or an operation (possibly the same one) to be invoked. Specific concurrency constraints can easily be added to this general framework. To elaborate on the interleaving behaviour we would wish to define, we see two particular cases, namely interleaving of sequences at the top level, and an interleaving parallel operator $(||)$. Thus if we have two objects obj_1 and obj_2 and operation definitions: [1]

$$obj_1.op_1 \mathrel{\widehat{=}} a \mathbin{⨾} b \mathbin{⨾} c$$
$$obj_2.op_2 \mathrel{\widehat{=}} x \mathbin{⨾} y \mathbin{⨾} z$$

then interleaving of the sequence $a \mathbin{⨾} b \mathbin{⨾} c$ with the sequence $x \mathbin{⨾} y \mathbin{⨾} z$ is possible. Within a sequence we might also define a parallel sequence: $a \mathbin{⨾} (x \mathbin{⨾} y \mathbin{||} w \mathbin{⨾} z) \mathbin{⨾} b$ Where $x \mathbin{⨾} y$ and $w \mathbin{⨾} z$ can be interleaved with each other (and also with other sequences at the top level). This semantics allows us to talk about traces of behaviour, which can be used to relate actions to tasks [4] and to reason about the likelihood and consequences of sequencing errors [19].

2.4 Enhancement

An interesting result of this semantics, in relation to reasoning about systems is that it allows us to develop a more formal notation of enhancement. We can talk about possible traces of an operation, where the alphabet is the set of sequentially composed schemas. Consider the notion of *composition refinement* [13]; the definition of which states that one action is a composition refinement of another if its alphabet is a superset of the other, and if it is a subset of the other when projected onto its alphabet.

$$I \sqsubseteq_C J \Leftrightarrow (\alpha(I) \subseteq \alpha(J)) \wedge (traces(J) \upharpoonright \alpha(I) \subseteq traces(I))$$

Formalising the notion of *composition enhancement* introduced earlier:

$$I \sqsubseteq_{CE} J \Leftrightarrow (\alpha(I) \subseteq \alpha(J)) \wedge (traces(J) \upharpoonright \alpha(I) = traces(I))$$

[1] For consistency we use the symbols "$⨾$" and "$||$" for interleaved sequential and parallel composition.

We can define object based versions of the above, saying that an action is an enhancement of another if the set of objects which participate in it (denoted ω) are a superset of the other and if it is trace equivalent when projected onto the object set of the other (operator $\upharpoonright_{obj}$). A behavioural enhancement is necessarily a composition enhancement.

$$X \sqsubseteq_{be} Y \Leftrightarrow (\omega(X) \subseteq \omega(Y)) \wedge ((\mathit{traces}(Y) \upharpoonright_{obj} \omega(X) = \mathit{traces}(X))$$

These forms of enhancement help us to reason intuitively about changes to behaviour and dialogue of interactive systems specified as a collection of objects.

2.5 Reuse

The schema calculus and the evaluation model of Z can be applied in a straightforward fashion to arrive at a semantics similar to the action model. In Object Z, operations are 'selected' (by the environment) from the set of currently enabled operations, and data (input parameters) may be provided by the environment. This is similar to the action model, where both the action and the objects it executes with are selected by the environment. Hence a schema at the top level, which has a precondition or guard, selectable by the environment, and whose inputs are provided by the environment, can be thought of as an action. Operations defined at the top level may also be defined with schema expressions (expressions in the schema calculus), and in fact this must be done where inputs are provided by other schemas in the system. Since choice and selection are possible within schema expressions we can view a component schema as a form of action. This is strengthened by the interleaving semantics which allows interaction between schemas through the state transitions they perform.

The fact that operations can themselves be constructed from other schemas, by means of the schema calculus, makes it possible to deal with the composition of behaviour and communication/interconnection. Environment enrichment can be used to denote explicitly (possibly nondeterministic) selection by the environment.

Let us now consider the reuse criteria. This style can be criticised from the *contextualisation* viewpoint as being another form of a meta-method approach. In such an approach, higher level methods are used to call methods in given objects and define the manner in which they interact. However there are differences:

– the top level is still selected by the environment and as such the unit can be reasoned about as an action.
– the combination of interconnection and enablement allows the schemas themselves to be written with a relatively low degree of dependency. Thus even if the inputs are supplied by other schemas in the expression, we still write schemas as if they were being produced by the environment (ie. in the action style).
– there is a difference between a schema expression and what may appear in the body of a schema, particularly in the context of an interleaving semantics.

Looking at the other reuse criteria, the *dependency* aspect is good. This is because the source and destination of communication is entirely dependent on the context in which the schema appears, and hence unnecessary dependency has not been introduced to the communication. Also, schemas have preconditions allowing dependencies to be encoded at a relatively high level of abstraction. Likewise for the *viscosity* issue, precondition schemas allow us to alter inter-object behaviour without modifying schema bodies.

3 Example

A small case study of a web browser is presented below to illustrate the style of specification we envision. We start by noting the main components of functionality the application is to provide. File access and markup form the functional core, file presentation, hyperlinks and navigation form the user interface component. We have a number of requirements on the interaction the system is to support. The user must be able to: (1) move within the text (2) resize text (3) select hyperlinks (4) type in a URL. We opt to distinguish interaction with text (especially hyperlinks) (requirements 1-3) from rest of front end (requirement 4). The interface is defined with two interactors (HText and Location) and the system with a single *Core* class, each of which is specified as an Object-Z class. The *HText* interactor represents the component displaying the page, and so includes operations to move within the page[2], resize the page display and to display a new page.

$HText$

$contents : tagged;$
$links : \mathbb{P}\ linktype;$
$position : \mathbb{N};$
$size : sizetype$

$position \in \operatorname{dom} contents$

new_text
$\Delta(contents)$
$text? : tagged$

$move\_up$
$\Delta(position)$

$move\_down$
$\Delta(position)$

$resize$
$\Delta(size)$
$newsize? : sizetype$

The *location* interactor simply displays the address of the page currently being displayed:

$Location$

$loc : URL$

new_loc
$\Delta(loc)$
$url? : URL$

We must also identify the operations which the functional core will provide, namely the retrieval of files (from a URL) and markup of HTML files into tagged text.

$Core$

$getfile$
$url? : URL;$
$url! : URL;$
$file! : HTML$

$markupfile$
$file? : HTML;$
$text! : tagged$

[2] Schema bodies are omitted for brevity.

We use a higher level class to tie the specification together [3], and to contain joint behaviour:

$System$

$\upharpoonright(hlink, type\_url, move\_up, move\_down, resize)$

$text : HText;$
$site : Location;$
$fcore : Core$

[Composite operations to be defined here]

The first item of joint behaviour performs file fetching, markup and display, taking a URL as input, and takes the form of a sequential composition.

$$new\_url_0 \mathrel{\widehat=} fcore.getfile \mathbin{;} fcore.markupfile \mathbin{;} text.new\_text$$

Then we can specify the actions to handle clicking on hyperlinks and typing locations directly using nondeterministic selection (by the user), from the set of links within the current page ($text.links$), with the joint behaviour defined above (see fig. 1):

$$hlink \mathrel{\widehat=} [\, url? : text.links \,] \bullet new\_url_0 \mathbin{;} site.new\_loc$$
$$type\_url \mathrel{\widehat=} [\, url? : URL \,] \bullet new\_url_0$$

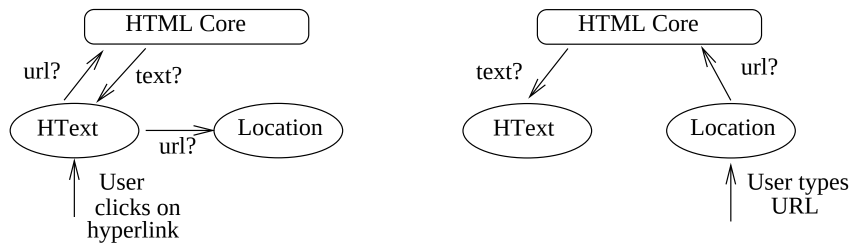

Fig. 1. *hlink* and *type-url* behaviours

Enhancing behaviour We will develop our specification by means of two enhancements, the first of which is a simple history mechanism, by means of which the user can move through the list of previously visited sites. In terms of our dialogue, we can allow for this by means of buttons which move the user either one step forwards or backwards in the list of previously visited sites. A single interactor should suffice for this purpose.

$History$

$fw : \mathrm{seq}\ URL;$
$bw : \mathrm{seq}\ URL$

new_site
$\Delta(fw, bw)$
$url? : URL$

$fwpress$
$url! : URL$

$bwpress$
$url! : URL$

$fwhist$
$\Delta(fw, bw)$

$bwhist$
$\Delta(fw, bw)$

[3] The $\upharpoonright$ expression lists the visible operations of the class.

Selecting a new site from a previously visited site will discard sites in the history after the current one. An instance of the history must obviously be defined[4]:

$$hist : History$$

The *new-site* action must be invoked whenever a new site is visited and so we must define a new joint behaviour:

$$new\_url_1 \mathrel{\widehat=} fcore.getfile \mathbin{⨾} fcore.markupfile \mathbin{⨾} text.new\_text \mathbin{⨾} hist.new\_site$$

The new_url_1 operation is a behavioural enhancement of new_url_0, as the same set of possible traces are yielded when the traces of new_url_1 are projected onto the object set of new_url_0:

$$new\_url_0 \sqsubseteq_{be} new\_url_1$$

Note that with the event model we would have had to alter one of the interactors themselves with this configuration specific detail. We also see that Griffiths' [10] composition semantics (described later), which does not hide matching inputs and outputs allows us to make such additions without altering the other schemas to 'preserve' the *url* input. The action specifications for *forward* and *backward* are quite straightforward.

$$forward \mathrel{\widehat=} hist.fwpress \mathbin{⨾} new\_url_0 \mathbin{⨾} hist.fwhist$$
$$backward \mathrel{\widehat=} hist.bwpress \mathbin{⨾} new\_url_0 \mathbin{⨾} hist.bwhist$$

Enhancing Dialogue Another interesting aspect is the way we can refine these dialogue definitions to include error handling. Consider the *new-url* action and the two possible failures:

1. unable to obtain file (*getfile* fails)
2. bad file syntax (*markupfile* fails)

In an action context, handling these errors would involve strengthening the guard on *markupfile* (to determine whether *getfile* has completed successfully) and on *new_txt* (to determine whether *markupfile* has completed successfully). In the schema calculus, this corresponds to adding a "precondition schema" which constrains it's input variables (and does not manipulate state) through sequential composition (or conjunction), for example: $ok \mathrel{\widehat=} [\, status? : \mathbb{B} \mid status? \,]$. Expressions including *markupfile* and *new_text* can then be prefixed with this precondition schema. This allows us to deal with these failures quite cleanly in the behaviour definition, which uses the choice operator:

$$\begin{aligned}
new\_url_2 \mathrel{\widehat=}\ & fcore.getfile \mathbin{⨾} \\
& \quad ((ok \mathbin{⨾} fcore.markupfile \mathbin{⨾} \\
& \quad\quad ((ok \mathbin{⨾} text.new\_text \mathbin{⨾} hist.new\_site) \\
& \quad\quad\ [] \ (notok \mathbin{⨾} markup\_error))) \\
& \quad\ [] \ (notok \mathbin{⨾} getfile\_error))
\end{aligned}$$

It can be seen that the new_url_2 operation is a compositional enhancement of new_url_1 due to the prefix closure property of the *traces* function. The capabilities of the interleaving semantics can be exploited when dealing with concurrent operations the specification includes the possibility of performing operations such as *resize* while a *new-url*

[4] In fact, we could use mechanisms such as inheritance to perform such state extensions.

action is in progress. Furthermore, the effects of the resize can be taken into account in the markup;

$$new\_url_3 \mathrel{\widehat=} fcore.getfile \mathbin{\raise 0.6ex\hbox{\oalign{\hfil$\scriptscriptstyle \mathrm{o}$\hfil\cr\hfil$\scriptscriptstyle \mathrm{9}$\hfil}}} [\, dsize! : sizetype \mid dsize! = text.size \,] \mathbin{\raise 0.6ex\hbox{\oalign{\hfil$\scriptscriptstyle \mathrm{o}$\hfil\cr\hfil$\scriptscriptstyle \mathrm{9}$\hfil}}} fcore.markupfile$$

Other examples could include specification of behaviour for an 'abort' function which would halt a *new-url* operation already in progress.

Discussion A number of the issues discussed earlier may be seen in the above examples. Returning first to our characterisation of the reuse problem:

– Precondition schemas such as ok have been used in a manner similar to the strengthening of guards in the action model. This has allowed us to make changes to inter-object behaviour (such as the conditions under which *markupfile* can occur) without altering the operation schemas.
– The schema calculus has also allowed resolution of the *dependency* issue, since the inputs and outputs of an operation are not associated with a particular source or destination. Thus in the case of *markupfile* the operations output can determine the next operation, but this is not encoded into the operation itself.
– *Contextualisation* remains something of a problem, but is alleviated by concepts such as environmental selection and the behaviour of the choice operator within the interleaving semantics.

The atomicity of sequentially composed schemas also allows us to talk of traces of such schemas within composite operations, making a more rigorous treatment of enhancement of behaviour and dialogue possible.

4 An Overview of the Formal Semantics

In this section we shall summarise a semantics embodying interleaving behaviour. Our semantics is based on that given by Griffiths [10]. Only those aspects of the semantics which differ from the original are detailed, and we shall endeavour to communicate how each component relates to the overall semantic architecture. A diagram of the relationship between components of the semantics is given in figure 2. The aim of the

$$\begin{tikzcd}
 & \text{Composite Operations} \arrow[r, "\textit{meaning fns}"] \arrow[dl, "\textit{message effects}"'] & \text{seq En\_op\_val} \arrow[dr, "\textit{attribute of}"] & \\
\text{Effect} \arrow[dr] & \text{Class\_meaning} & \text{Object\_val} \arrow[l, "\textit{objects}"'] & \text{Class\_struct} \arrow[l] \arrow[dl, "\textit{attribute of}"] \\
 & \textbf{Spec\_meaning} & \text{Spec\_struct} \arrow[l, "\textit{transitions}"'] &
\end{tikzcd}$$

Fig. 2. Semantic Architecture

semantics is to formalise the notion of interleaving and interleaving parallel detailed earlier.

Two-part model The basic premise of Griffiths' semantics [10] is that all operations can be reduced to a simple two-part form, consisting of an internal transition and an external message. Griffiths defines a syntactic translation from 'full' Object-Z to this sub-language; since we use the same sub-language we need not include the details here. The two-part simple operation is the basic building block of the sub-language. Each operation is represented by a set of *enabled operation values* or *En_op_val*, which consist of a single state in which the operation is defined, and a set of *transactions*. A transaction is a relation between an $(in, post, out)$ triple describing the **internal transition**, and a *message* describing **external interaction**. The pairs in the transaction relation capture the behaviour under different inputs and the possibility of non-determinism. Each *En_op_val* thus encodes all the possible transitions from a given state.

Composition Operators The first level of the semantics relates simple operations to composite operations. Since the meaning of an operation is represented by a set of *En_op_val*, the composition operators are defined by a set of meaning functions mapping sets of *En_op_val* representing the composed schemas to a single set of *En_op_val* representing the composite operation:

$\mathcal{M}_{sequence} : (\mathbb{P}\, En\_op\_val \times \mathbb{P}\, En\_op\_val) \rightarrow \mathbb{P}\, En\_op\_val$

This must be altered to accommodate the notion of interleaving; particularly we must keep the components of sequences distinct in order to allow an interleaving semantics. In our semantics, all these functions manipulate sets of *sequences* of *En_op_val*. A simple operation is represented by a set of sequences of length 1. In this way, the meaning of any schema expression is defined by a set of sequences of *En_op_val*. We also introduce a meaning function for the interleaving parallel operator:

$\mathcal{M}_{parallel} : (\mathbb{P}\,\mathrm{seq}\, En\_op\_val \times \mathbb{P}\,\mathrm{seq}\, En\_op\_val) \rightarrow \mathbb{P}\,\mathrm{seq}\, En\_op\_val$

Class, Object and Specification Meaning The next level of the semantics relates operation definitions in the form of sets of sequences of *En_op_val* to the meanings of classes and hence to the meaning of a specification. With the redefinition of the meaning of an operation as a sequence, the class and object meanings must be altered. In the definition of *Class_struct*, the *ops* attribute containing the operations of the class now maps names to sequences of *En_op_val*; $ops : Name \mathrel{+\!\!\!\!\twoheadrightarrow} (\mathbb{P}\,\mathrm{seq}\, En\_op\_val)$. The state of an object at a particular point in time is represented by an *Object_val*, which is extended to encompass the state *between* steps of a sequence. This is critical since the meaning of a class is defined in terms of object values and transitions between them. So the *Object_val* class is extended with an attribute which contains the sequences of *En_op_val* for both the full and partially executed operations, and represents the set of possible sequences of transitions from the current object value. This set directly supports our new evaluation model; object transitions are extracted from this set rather than from the *ops* attribute (as in the original semantics).

Class meaning is interpreted from class structure by means of three functions: *objects* which produces the set of well formed values, *initial_objects* which does likewise for the initial states, and *OT*, which produces the set of object transitions defined by the class. These functions are rewritten to incorporate the new definition of object value, and the representation of operations by sets of sequences of *En_op_val*. Both the objects and transition map may include unreachable values, so the functions *reachable_objects* and *poss_ob_behavs* are defined as those reachable from the initial states. Since OT now includes all "internal" transitions, *reachable_objects* and *poss_ob_behavs* are as in [10].

Relating Messages to Effects The final level of the semantics concerns the *effect* of a given message on an object map under a particular set of inputs. This aspect is closely related to the first (compositional) component of the semantics. Both are required to fully define the behaviour of a specification. The *Effect* structure comprises two object mappings (before and after), a message, and valuations for the input and output parameters [10].

$$Map == O \nrightarrow Object\_val$$
$$Effect == Map \times Map \times Message\_val \times Valuation \times Valuation$$

The effect of composite messages are constructed from the "known" effects of the component messages. An interleaving parallel message type is introduced which denotes that the messages composed are to be combined in an interleaved parallel fashion. This must be done since we wish to define an effect set for operations which are occuring at the same time, a concept not previously defined in the schema calculus. The effect set is then used to determine the system transitions, and hence the *meaning* of a specification (see [10] for more details).

5 Discussion

To discuss this approach to the specification of joint behaviour, it is appropriate to return to our original concerns. The interactive systems issues have been addressed through the use of a single consistent notation, which allows us to reason about behaviour and dialogue. This has allowed us to avoid the attendant problems of hybrid notations. We have done this within the framework of a model based object oriented specification language (Object Z) and thus are in a position to apply object oriented techniques to specification and reuse.

We have seen that performance of the schema calculus with respect to the reuse problem was good. When specifying an operation with a certain precondition and input parameters, no assumptions are made about how these preconditions are met and parameters supplied, which helps reduce *dependency*. The schema calculus allows us to conjoin schemas with existing schemas to achieve strengthening of guards and state extension as is the case with the action model. This allows us to reuse schemas which include object interaction, without modification, in a range of different contexts, and is a significant step towards addressing the *viscosity* issue. *Contextualisation* is more problematic, although there are similarities between the evaluation model of Z and the action model. It is still an object only system, albeit one in which action definitions are easily constructed within a given object by means of the schema calculus. So in this case one might say that the problem has been reduced in scale and severity, but not completely eliminated.

In summary, our aim has been to integrate descriptions of joint behaviour and dialogue in the context of object oriented specification of interactive systems. Our approach has been to use a single notation, taking into account interactive systems concerns such as the analysis of enhancements and more general concerns such as reuse and expressive power.

References

1. G. D. Abowd. *Formal Aspects of Human-Computer Interaction*. DPhil Thesis, University of Oxford Computing Laboratory: Programming Research Group. 1991. Available as Technical Monograph PRG-97.
2. H. Alexander. Structuring dialogues using CSP. In M. D. Harrison and H. W. Thimbleby, editors, *Formal Methods in Human Computer Interaction*, pages 273–295. Cambridge University Press, 1990.
3. C.J. Bramwell. *Formal aspects of the Design Rationale of Interactive Systems*. DPhil Thesis, Department of Computer Science, University of York. 1995.
4. A.M. Dearden and M.D. Harrison. Formalising human error resistance and human error tolerance. In *Proceedings of the Fifth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace*. EURISCO, 1995.
5. G. Doherty and M.D. Harrison. Reuse in action and event based object oriented systems. Department of Computer Science, University of York, 1998.
6. D. J. Duke and M.D. Harrison. Abstract interaction objects. *Proceedings of Eurographics '93*, Computer Graphics Forum, 12(3), 1993.
7. D.J. Duke and M.D. Harrison. Event model of human-system interaction. *Software Engineering Journal*, 10(1):3–12, 1995.
8. R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Computer Standards & Interfaces*, 17:511–533, 1995.
9. B. Fields, N. Merriam, and A. Dearden. DMVIS: Design, Modelling and Validation of Interactive Systems. In M.D. Harrison and J.C. Torres, editors, *Proceedings, 4th Eurographics Workshop on Design, Specification, and Verification of Interactive Systems*, Springer Computer Science. Springer Wien, 1997.
10. A. Griffiths. Object-oriented operations have two parts. In D.J. Duke and A.S. Evans, editors, *Proceedings of BCS-FACS Northern Formal Methods Workshop, Ilkley*, 1997.
11. H-M. Järvinen. *The design of a specification language for reactive systems*. Doctor of technology, Tampere University of Technology, 1992. Available as Tampere University of Technology Report 95.
12. Jifeng He. Various simulations and refinements. In *Volume 430 of Lecture Notes in Computer Science*, pages 340–360. Springer Verlag, 1989.
13. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.
14. A. Hussey and D. Carrington. Using Object-Z to specify a web browser interface. In J. Grundy and M. Apperley, editors, *OzCHI '96 – The Sixth Australian Conference on Computer-Human Interaction*, pages 236–243. IEEE Computer Society Press, 1996.
15. C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, 1986.
16. Y. Joung and S. Smolka. A comprehensive study of the complexity of multiparty interaction. *Journal of the ACM*, 43(1):75–115, January 1996.
17. H. Mili, F. Mili, and A. Mili. Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering*, 21(6), June 1995.
18. J.M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1988.
19. P. Wright, B. Fields, and M. Harrison. Deriving human-error tolerance requirements from task analysis. In *Proceedings, ICRE'94 The First International Conference on Requirements Engineering, Colorado Springs*, pages 135–142. IEEE, April 1994.