On Modelling and Analysis of Dynamic Reconfiguration of Dependable Real-Time Systems

Manuel Mazzara and Anirban Bhattacharyya
Reconfiguration Interest Group
School of Computing Science, Newcastle University
Newcastle upon Tyne, UK
Email: {*Manuel.Mazzara, Anirban.Bhattacharyya*}*@ncl.ac.uk*

Abstract—**This paper motivates the need for a formalism for the modelling and analysis of dynamic reconfiguration of dependable real-time systems. We present requirements that the formalism must meet, and use these to evaluate well-established formalisms and two process algebras that we have been developing.**

Keywords-**Requirements, dynamic reconfiguration, modelling, analysis, verification**

I. Introduction

Modern dependable real-time (DRT) systems are required to have greater flexibility, availability and dependability than their predecessors. One way of achieving this is through the use of dynamic reconfiguration techniques. A system can be implemented as a collection of configurations, where each configuration is a network of communicating components. Different configurations can be optimized for different services or operating conditions. Therefore, facilities for replacing one configuration by another at runtime can increase the flexibility of the system. Runtime facilities for creating configurations that provide new services, and for removing configurations providing services that are no longer required, can reduce the downtime for maintenance; thereby increasing the availability of the system. Replacing an executing configuration when it exhibits erroneous behaviour with a valid configuration increases the system's reliability; and thereby increases its dependability. However, dependability is a combination of reliability and predictability, and ensuring predictability during dynamic reconfiguration is difficult. The main reason for the difficulty is the interaction between the application services that the system is required to provide to its environment and the reconfiguration services that the system must also perform in order to support its flexibility, availability and reliability. The interaction between the two kinds of service implies that neither can be analyzed in isolation.

Existing research on DRT systems has largely concentrated on system design and programming languages (see [15] and [9]) rather than on formalisms and their methods. However, formal methods are important because they are useful in providing the strong guarantee of system correctness required for such systems. Furthermore, the formal research on DRT systems has focused on scheduling (see [32], [5] and [26]) rather than on computational models. As for the existing research on dynamic reconfiguration, it has either assumed mode changes to be instantaneous or has implicitly assumed the controlled environment can wait whilst the control system is reconfigured [16]. Both assumptions are unrealistic for DRT systems. For example, it is impossible to perform an instantaneous mode change in a distributed control system because a distributed system has no global state; and suspending or aborting application services during reconfiguration in an unstable 'fly-by-wire' aircraft would cause the aircraft to become unstable, and possibly suffer catastrophic failure. Thus, there is very little research on computational models with overlapping modes – the most appropriate form of dynamic reconfiguration for DRT systems. Therefore, the purpose of our research is to develop a computational formalism for DRT systems, in which interactions between application and runtime system activities can be modelled; and the model can be used to verify safety and liveness requirements of the system. DRT systems are typically used to maintain the stability of unstable environments and to keep them under control. Therefore, they are characterised by time-critical concurrently executing activities with hard deadlines, small synchronisation tolerances between events, and tight resource constraints. Hence, our formalism must be able to express both reconfiguration and real-time features of DRT systems, and must be able to verify both their functional and timeliness properties.

This paper makes three contribution. First, it identifies requirements on a formalism for DRT systems that have overlapping modes. Second, it evaluates well-established formalisms against these requirements. Third, it briefly describes two novel process algebras that we believe progress the state of the art in this field. The rest of the paper is organized as follows: section II describes a simple framework that illustrates the scope of the reconfiguration we are considering, and then identifies the structural, modelling and analysis requirements that a formalism targeted on the dynamic reconfiguration of DRT systems must meet. Section III uses these requirements to evaluate well-established

formalisms and identifies their strengths and weaknesses. Sections IV and V describe two formalisms we have been developing, and evaluates them against the requirements.

II. REQUIREMENTS ON A FORMALISM FOR DYNAMIC RECONFIGURATION

Reconfiguration of a DRT system typically involves changing one configuration – a network of communicating and concurrently executing components of the system – into another. Software components can also migrate between networked computers. The computers and their network constitute the hardware platform of the system, which does not change. We can represent the system using a simple framework (see figure 1). The *application layer* consists of those components of the system and their communication connectors (i.e. *objects* and *links*) that are the focus of reconfiguration. The *location layer* (which does not change) is used to represent those components and connectors (i.e. *nodes* and *channels*) that are necessary in order to describe the migration of objects and links. Thus, reconfiguration can be represented as the creation and deletion of objects and links, and as changes in the mappings between objects and links, objects and nodes, links and channels, and links and nodes.

It is important to notice that a method for reconfiguration is not included in the framework or in the requirements. This is because we believe that determining a method is the responsibility of the system designer, rather than the formalist. As an analogy, the differential and integral calculus does not contain a method for designing car engines; but the calculus is still useful to this regard. These aspects have been properly discussed in [22].

In order to be practicable, it must be possible to support the formalism with tools for both modelling and automated analysis.

The requirements we have identified fall into three categories: structure, modelling and analysis. The completeness of these categories can only be determined with respect to a case study, which is beyond the scope of this paper.

Structural Requirements

It must be possible to model components in a recursive and compositional way. To do this, models must be organizable in terms of units, with each component of a system expressible as a composition of one or more units, and each unit corresponding to only one component. This way, it should be possible to express independent reconfiguration of units, in the same way that components can be independently reconfigured.

Modelling Requirements

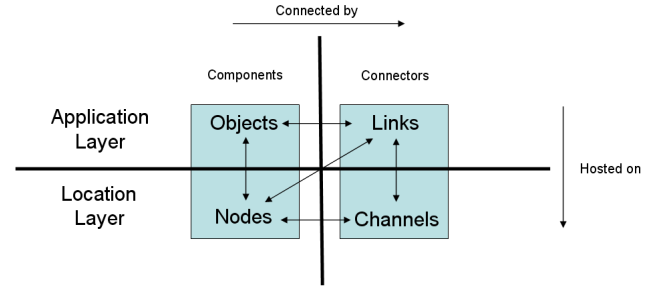The formalism must be able to express the following:

Figure 1.   Framework for Dynamic Reconfiguration

1) Reconfiguration of Components: it must be possible to express the creation, deletion and replacement of objects, and also the migration of objects between nodes (see figure 1).

2) Reconfiguration of Connectors: it must be possible to express the creation and deletion of links between objects.

3) Application Behaviour: it must be possible to express the functionality of an application in terms of basic activities (such I/O, data transformation and data manipulation) and their composition using sequencing, parallelism, alternatives, iteration and recursion.

4) Interference with Runtime Support: it must be possible to express interference between the application and reconfiguration activities of the system. The interference can be functional or temporal, and can occur in both directions. For example, functional interference can occur because the replacement of an object can change the output of the application; and conversely, the execution of an object that is to be replaced can change its state, and the reconfiguration activity must take this state change into account when replacing the object. Temporal interference is the effect on timing properties (such as execution time) of the concurrent execution of application and reconfiguration activities; and it can occur in the absence of functional interference.

5) Real-Time Information: it must be possible to model concurrent activities in terms of their ordering, duration, communication, interrupts and memory usage for schedulability analysis. Clocks must also be modelled.

6) Real-Time Restrictions: it must be possible to express restrictions arising from requirements or resource limitations, such as deadlines, computation times and bounds on synchronization. Since memory is limited, it must be possible to express restrictions on memory usage.

7) Fault Tolerant Behaviour: it must be possible to express the failure modes of a system and their failure rates; and the way in which errors will be recovered.

Analysis Requirements

The main purpose of modelling is analysis. Therefore, we identified a number of analyses that must be supported by the formalism: the formalism must be consistent. That

is, it must not be possible to prove both a statement and its negation; otherwise, the result of an analysis can be logically invalid. Critical properties of the system modelled using the formalism must be decidable. That is, the function evaluating these properties must be Turing computable [33]. If these two requirements are not met, it will not be possible to provide automated tool support and the formalism will not be used in practice.

Termination of critical activities must be decidable (see 'Analysis Requirements' in section IV for a discussion). That is, it must be possible to decide whether or not any activity will stop its execution. Deadlock detection must also be decidable. That is, it must be possible to decide whether or not two or more processes can wait indefinitely for each other in a cycle [36]. It must also be possible to decide whether or not activities can meet their deadlines with respect to a scheduling discipline. It must be possible to check whether or not data transformation and date manipulation are type correct. It must be possible to perform reliability analysis using the failure modes and failure rates given in the model.

III. EVALUATION OF FORMALISMS

The requirements described above can be used to evaluate existing formalisms for their suitability for the modelling and analysis of dynamic reconfiguration of DRT systems. Lack of space prevents a detailed and comprehensive review. Therefore, we briefly review a selection of well-established model-based formalisms, process algebras, Petri nets and other formalisms, and identify their significant strengths and weaknesses. Notice that the parts dedicated to "Analysis Requirements" are more focused on the tool support existing for the specific formalism, since some of the properties described above are not always applicable (e.g. deadlock for sequential formalisms) or are obviously satisfied.

A. Model-based formalisms

There is a long tradition of methods falling into this category. The most famous are probably VDM [8], Z [3], B-method [1] and, lately, Event-B [2]. The mathematics underlying these formalisms are set theory and first order logic. The approach consists of modeling the system's state in terms of sets and functions, and modelling state transformation using operations (or events in the case of Event-B). Predicates are used to express invariant conditions on the state. In Z, the emphasis is on formal specification, whilst with the B-method emphasizes the "method" itself. Both B and Event-B focus on the application of stepwise refinement (reification in VDM). That is, the verifiable transformation of an high-level formal specification into an executable program.

Structural Requirements

The VDM Specification Language (VDM-SL) and its extended form (VDM++) deal with structure in different ways: VDM-SL uses modules whilst VDM++ (being object-oriented) uses classes with multiple inheritance. Thus, both can express structural information. Classical B and Event-B specifications are instead organized in machines that also impose a basic structure.

Modelling Requirements

Reconfiguration is not natively supported by these formalisms, but it can be encoded with difficulty (as in Turing Machines). None of these formalism has been designed for reconfiguration. Also, real-time information and restrictions are not natively supported. Temporal extensions exist (e.g. for Z), but their use of temporal logic does not allow to meet all the real-time requirements. Fault tolerance is not natively supported, but many contributions exist in this field. The Deploy project (www.deploy-project.eu) is addressing these issues for Event-B.

Analysis Requirements

Model-based formalisms are mature, and they tend to have extensive tool support. For example, Overture for VDM (www.overturetool.org) and Rodin for Event-B (www.event-b.org). They have facilities for type checking, verifying partial correctness of a design, and checking termination. However, they are more targeted on sequential systems, and (therefore) properties characteristic of concurrent systems, such as deadlock freedom, are not directly addressed. This does not mean that they cannot be used to model concurrent systems at all. In particular, Event-B can represent interactive systems. However, events are atomic and are associated with an interleaving semantics without interference.

B. Process Algebras

While the formalisms described so far are mainly concerned with functional properties and sequential behavior, process algebras are about interaction between concurrent processes. Among the original methods in this field, we can mention CSP [13] and CCS [24]. Mobile process algebras (e.g. Milner's π-calculus [25]) represent a further development addressing mobility.

Structural Requirements

The common structural unit of all process algebras is a communicating concurrent process. Process algebras supporting a basic form of structuring do exist, although of different nature if evaluated with respect to the requirements discussed in this paper. For example, the ambient calculus [10] includes a notion of locations and mobility. Webπ_∞ is described later in this paper with the structure it imposes on the π-calculus.

Modelling Requirements

Mobile process algebras like the π-calculus are interesting because of their treatment of component bindings as first class objects, which enables link reconfiguration to be expressed simply. Although proper component reconfiguration is absent, the reconfiguration mechanism on which the π-calculus is built already represents a seminal form of what is described in this paper. Extensions to support real-time and fault tolerance are an active area of research.

Analysis Requirements

The weakness of this category of language is tool support. Although different bisimulations have been defined and tailored to specific needs, tool support is still limited. It is worth mentioning TyPiCal, a type-based static analyzer for the π-Calculus [14]. TyPiCal is able to provide four different kinds of program analyses and transformations: lock-freedom analysis (certain communications or synchronizations will eventually succeed), deadlock-freedom analysis, useless-code elimination (it removes sub-processes that do not affect the observable behavior of the process), and information flow analysis. The type system is extended in such a way that channel types carry information on how channels are used. This allows a type inferencer to obtain information about the behavior of a process. As a drawback, the expressive power of the type system is limited.

C. Petri Nets

Petri nets [28] are a graph-based formalism to represent concurrency. They are a mathematical formalism, but they also come with an appealing graphical notation in the style of UML activity diagrams [12]. To the best of our knowledge, Petri nets were the first formalism for describing concurrency. A formal account of Petri nets in the form of a survey can be found in [20].

Structural Requirements

Structural information like modules is not natively expressible in Petri nets. Although it is a suitable formalism to express parallel and distributed systems, for a long time it did not fully support compositionality, and this deficiency prevented its wide use in large real-world applications. The recent 'hype' on formalisms for verification of Web Services composition lead to some work done in this field in opposition to the process algebra approach. Indeed a big debate arose in the recent years to this regard and it is well explained in [34] and in the conclusions of [19]. Some work for enhancing Petri nets compositionality in other contexts has been done (a survey in [4]) and also work on modularization and Petri nets do exist [27] but we are not aware of applications.

Modelling Requirements

Petri nets do not offer a native way for addressing dynamic reconfiguration, but extensions to the formalism have been presented to allow for an easy formalization of this feature. For example, reconfigurable Petri nets [18] are a subclass of net rewriting systems with the goal of enhancing the expressiveness of the basic model. Other approaches to model dynamic reconfiguration have been tried and shown through case studies [17]. The original version of Petri nets is not Turing complete but extension have been provided later to add expressiveness.

Analysis Requirements

Tool support for Petri nets benefited from decades of research on the topic (see www.informatik.uni-hamburg.de/TGI). Various kinds of Petri net are supported by tools, and many of these tools offer a practical and appealing graphical editor not offered by other formalisms. Animation and model checking are other useful features offered by some of these tools. Overall, it is probably not an exaggeration to say that, in comparison to process algebras, tool support for Petri nets have received much more attention from the scientific community.

D. Other Formalisms

We are aware of others formalisms that deserve attention. Here we will briefly mention some of those with their main features.

The Chemical Abstract Machine (CHAM) [6] exploits the chemical metaphor. That is, it follows an approach based on viewing software systems as chemicals whose reactions are rigorously controlled by specific rules. The original idea was to bridge the gap between Petri nets, which can be considered as abstract machines but lack expressiveness (in the basic version) and process algebras, which are more expressive but are intended as specification formalisms for distributed systems (rather than as abstract machines). It is also worth mentioning how the gap between calculi for concurrent processes and languages for programming distributed and mobile systems is not really bridged by CCS or π-calculus, since their interactions are based on rendezvous, i.e. atomic and non-local, which is hard to implement fully in a distributed setting. Here is where CHAM finds its niche and it is very interesting for its ability to represent a system as a syntactic description of the static components, the molecules, and of a set of reaction rules describing how the system evolves dynamically. This already shows how a system is structured in the formalism and how some form of reconfiguration can be expressed. Furthermore, formal reasoning, for example about deadlocks, can be performed.

Another attempt of combining pros and cons of different formalisms and bridging the gap between them is CSP‖B [30]. The authors recognize that a system can be projected into two different dimensions, the dynamic view and the

state view, and these projections have to show consistency. Thus, they combine B with a *controller language* able to *drive* a B machine and this controller language is (a subset of) CSP. In this way B is able to express requirements on the state of a system while CSP expresses the interactive behaviour. This approach should permit the exploitation of existing tool support for both CSP and B. CSP‖B it is interesting for the purpose of this paper since it combines the structure of B and it has the potential of exploiting mobility like the π-calculus, although not in the basic version [31].

For what concerns tool support, it is worth mentioning UPPAAL, an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types (www.uppaal.com).

IV. $Web\pi_\infty$

In [21] and [23] a unifying theory has been developed with the pragmatic intention of using it for encoding orchestration languages behavior (WS-BPEL in particular) and verifying process equivalence. Certainly, the reconfigurability needs in that scenario are limited compared to the general case: processes can be rolled-back or compensated, fault handlers activated, but they cannot be dynamically deleted or created. Both in WS-BPEL and in the developed theory $Web\pi_\infty$ there is always the need of statically defining a syntactical proximity between the process responsible for the normal behavior and the one responsible for the 'abnormal' one. This means that the two processes have to be statically bound. More complex behavior can be certainly encoded and the problem circumvented but we still have noticed that this practice would represent a sort of unpleasant 'hacking' that it is still far from what we want to offer to the final user. Despite this, we still recognize one step forward with respect to previous formalisms for what concern inborn reconfigurable behavior. First, being $Web\pi_\infty$ based on the π-calculus, it allows the same sort of flexibility of its ancestors, i.e. link passing. But it does more offering a reconfigurable behavior as a first class citizen. In fact, when compared to the ideal formalism — the cornucopia able to elegantly satisfy all the presented requirements — $Web\pi_\infty$ appears still primitive but it contains, in a seminal form, many of those requirements.

$Web\pi_\infty$ is a conservative extension of the π-calculus where the Workunit operator $\langle\!\langle P \ ; \ Q \rangle\!\rangle_x$ has been added. Here the normal behavior is expressed by the process P and the abnormal one by Q, while x is the "trigger" that allows to switch from one to the other. This "trigger" is able to activate Q during the execution of P if another parallel process $\overline{x}$ (output) requires it. The evolution of the parallel composition of an output and a Workunit (according to the formal reduction semantics) is: $\overline{x} \mid \langle\!\langle P \ ; \ Q \rangle\!\rangle_x \rightarrow \langle\!\langle Q \ ; \ \mathbf{0} \rangle\!\rangle$ that, for technical reasons here omitted, will then behave like Q.

Let us consider a simple example to show how $Web\pi_\infty$ can be used for dynamic reconfiguration. Let R be a reconfiguring process consisting of a body S. The body S behaves nondeterministically – either correctly or erroneously. In the first case, it will end up executing S itself (after 'garbage collection' of pending messages), while in the second it will send a message $\overline{e'}$ (again after "garbage collection"). We define S as follows: $S \triangleq v().e().S + e().v().\overline{e'}$. In the erroneous case, the handler of the Workunit in the reconfiguring process R will be activated restoring S itself than, in turn, can still behaves correctly or erroneously. We define R as follows: $R \triangleq \langle\!\langle S \ ; \ S \rangle\!\rangle_{e'} \mid \overline{v} \mid \overline{e}$. Let us show the reductions in more detail in both the normal and abnormal cases:

Normal case: $\langle\!\langle v().e().S + e().v().\overline{e'} \ ; \ S \rangle\!\rangle_{e'} \mid \overline{v} \mid \overline{e}$
$\rightarrow \langle\!\langle e().S \ ; \ S \rangle\!\rangle_{e'} \mid \overline{e} \rightarrow \langle\!\langle S \ ; \ S \rangle\!\rangle_{e'}$

Abnormal case: $\langle\!\langle v().e().S + e().v().\overline{e'} \ ; \ S \rangle\!\rangle_{e'} \mid \overline{v} \mid \overline{e}$
$\rightarrow \langle\!\langle v().\overline{e'} \ ; \ S \rangle\!\rangle_{e'} \mid \overline{v} \rightarrow \langle\!\langle \overline{e'} \ ; \ S \rangle\!\rangle_{e'} \rightarrow \langle\!\langle S \ ; \ \mathbf{0} \rangle\!\rangle$

We believe this formalism shows some elegance and it is able to integrate structural, modelling and analysis requirements.

Structural Requirements

$Web\pi_\infty$ expresses information on the structure being the system organized in different workunits. The basic structural unit is a process, and workunits are used to perform reconfiguration representing what is being changed and the change. They can be recursive (nested) and compositional. Each of the workunit could, ideally, reside in a different host and could be compiled and linked separately. Although it is a very basic mechanism for structural information it represents an improvement in comparison to the π-calculus.

Modelling Requirements

1) Reconfiguration of Components: reconfiguration of components is possible through workunits and handler activation. In a $Web\pi_\infty$ process behaviour and reconfiguration are represented by P (application) and the interaction between $\overline{x}$ and the workunit, which produces the execution of the process Q "replacing" P (reconfiguration).

2) Reconfiguration of Links: reconfiguration of links is inherited by the π-calculus and its notion of mobility (which is a basic form of dynamic reconfiguration). The π-calculus looks interesting because of its treatment of component bindings as first class objects, which enables dynamic reconfiguration to be expressed simply.

3) Application Behaviour: the functionality of an application in terms of basic activities are abstracted over in $Web\pi_\infty$. This feature is inherited from the π-calculus which has the purpose of representing process synchronization, communication and link mobility. Activities like sequence, parallel, alternative, iteration, etc. are expressible through encoding. This exercise has been done in [19] to encode

WS-BPEL. This shows how Webπ_∞ is expressive enough to be used to describe the application behavior.

4) Interference with Runtime Support: interference between the application and reconfiguration activities of the system is expressible through message passing and rendezvous, as shown in the example above. The interference can only be functional, not temporal.

5) Real-Time Information: it is possible to model concurrent activities in terms of their ordering and communication, but duration is not expressible (at least in the untimed version). Memory usage is not expressible, whilst interrupts are natively supported. Clocks cannot be modelled.

6) Real-Time Restrictions: Webπ_∞ is not able to express restrictions arising from time and space limitation limitations. However, a timed version of the language has been presented (but not coping with real time).

7) Fault Tolerant Behaviour: fault tolerance is a point in Webπ_∞ when describing, for example, the WS-BPEL recovery framework [11] and it has been one of the main reason for its development. Failure rates are not expressible.

Analysis Requirements

Webπ_∞ is a conservative extension of the π-calculus and can be encoded using it. The reason for developing a new formalism was not the nature of its expressiveness, but its pragmatics. As a consequence, consistency is inherited from the π-calculus. Termination, in the general case, is not decidable for Turing-complete formalisms, including π. In practical cases, and with the necessary restrictions (e.g. by typing and syntax), termination can be ensured. An adequate discussion on this topic is in [29]. The same holds for Webπ_∞. Decidability of termination is a theoretically well-known limit; when designing a formalism it is always a matter of practicality finding a compromise between the expressiveness of the languages and the properties that can be decided.

Looking at the other requirements, in Webπ_∞ it is not possible to decide whether or not the activities can meet their deadlines with respect to a scheduling discipline. It is not possible to check whether or not data transformation and manipulation is type correct. It is not possible to perform reliability analysis using failure modes and rates. Webπ_∞ comes with its tailored definition of bisimulation, i.e. a mathematical tool able to determine if two processes exhibit the same externally visible behavior. The proposed bisimulation is decidable for non-recursive processes and some properties are proved, as an example, in [21].

V. CCS^{dp}

CCS^{dp} is the first version of a formalism that is being developed specifically for the modelling and analysis of interactions between application and reconfiguration activities in DRT systems [7]. It is based on CCS [24], extended

with a single construct – the fraction process $\frac{P'}{P}$ – in order to reconfigure processes.

A fraction process $\frac{P'}{P}$ is used to replace and delete processes. On creation, the fraction $\frac{P'}{P}$ identifies any instance of a process matching the denominator process P with which it is composed in parallel, and replaces that process immediately and atomically with the numerator process P'. The matching can be either syntactic equality or a special kind of behavioural equivalence ($\sim_{of}$) we have defined. The reduction semantics is given by $P|\frac{P'}{P} \longrightarrow P'$ and $Q|\frac{P'}{P} \longrightarrow P'$ where $P \sim_{of} Q$. If no matching process instance exists, the fraction continues to exist until such a process is created (or the fraction is itself deleted or replaced). If there is more than one matching process instance, a non-deterministic choice is made as to which process is replaced. Similarly, if more than one fraction can replace a process instance, a non-deterministic choice is made as to which fraction replaces the process. Deletion of a process P is achieved by parallel composition with $\frac{0}{P}$. If P progresses to R, then $\frac{P'}{P}$ will not replace R by P' (unless R matches P). Notice that a fraction process has no intrinsic behaviour; it performs a transition only when composed with a process that matches its denominator.

We illustrate the use of fraction processes with a simple example. Let S be a server process that behaves either correctly (performing $\bar{v}$) or incorrectly (performing $\bar{e}$). Let R be a reconfiguring process that either lets S continue (when S behaves correctly), or replaces it with a different instance of the server process (when S behaves incorrectly). We can define S and R as follows: $S \triangleq \bar{v}.S + \bar{e}.S$ and $R \triangleq v.R + e.(\frac{S|R}{S})$

The reconfiguration of S is modelled by the transitions: $S|R \longrightarrow S|\frac{S|R}{S} \longrightarrow S|R$

The key strengths of CCS^{dp} regarding dynamic reconfiguration of DRT systems are: its ability to separate and compose models of application and reconfiguration activities, resulting in modular and terse models; expressing application and reconfiguration actions in the same form, so that their interleaving can be easily represented; and its simplicity in modelling process reconfiguration. The key weaknesses of CCS^{dp} are: its lack of facilities for naming processes, and for modelling real-time and fault tolerance properties of a system.

Structural Requirements

The basic structural unit is a process. The parallel composition operator ($|$) of CCS enables a process to be composed from parallel processes, and a process to be decomposed into parallel processes. The restriction operator (ν) facilitates process composition by scoping port names. These facilities are inherited by CCS^{dp}. The semantics of fraction processes enables application and reconfiguration activities to be modelled separately, and then composed using $|$ to enable reconfiguration to take place. Thus, fractions support

modular structuring of a model. However, since there is no naming scheme for processes, identical instances of a process cannot be selectively reconfigured.

Modelling Requirements

1) Reconfiguration of Components: replacement and deletion of components are expressed as reconfiguration transitions involving a fraction process. Component creation is expressed as process spawning (as in CCS). The granularity of process reconfiguration is a concurrent process, and any parallel composition of processes can be reconfigured. Process reconfiguration is atomic. Process migration cannot be modelled, since the location of a process is not represented.

2) Reconfiguration of Links: this can be expressed using process reconfiguration; but it is clumsy.

3) Application Behaviour: facilities for expressing this are inherited from CCS. A process can perform I/O actions (without value-passing), internal action, sequential action, iterative action, and make a deterministic or non-deterministic choice between alternative actions. Concurrent processes execute with interleaved semantics and communicate synchronously.

4) Interference with Runtime Support: this is expressed in terms of interleavings between the application transitions and the reconfiguration transitions of a process expression, and the resulting process expression shows the outcome of the interference. Temporal interference cannot be described, since CCS^{dp} has no time model.

5) Real-Time Information: this is inherited from CCS, and is limited. Concurrent processes and the order of actions can be expressed, but not the duration of an action (which excludes schedulability analysis). The synchronous communication between processes is also unsuitable for DRT systems. Memory usage, interrupts and clocks are not modelled.

6) Real-Time Restrictions: these cannot be expressed in CCS^{dp}.

7) Fault Tolerant Behaviour: there are no special facilities in CCS^{dp} to express this.

Analysis Requirements

Work on the analytical aspects of CCS^{dp} is in progress: proof of consistency is significant because of negative premises in some of the semantic rules of CCS^{dp}. Proof of decidability of the $\sim_{of}$ bisimulation is significant for matching. It is important to identify a practically useful set of processes for which we can prove decidability, termination and deadlock freedom. However, schedulability analysis, type checking and reliability analysis are beyond the scope of CCS^{dp}.

VI. CONCLUSION

In this paper, we have focused on reconfiguration with interference between application activities and reconfiguration activities in DRT systems. We have identified requirements on a computational formalism, and evaluated well-established formalisms (as well as our own) against these requirements. We have shown that none of the existing formalisms provides full native support for dynamic reconfiguration. Some implicitly contain specific features that can be useful when treating systems that inherently show reconfigurable features; but none of them are entirely suitable for this category of problems.

Other formalisms could also have been evaluated, but we decided to refer to Wermelinger's PhD thesis [35] which shares our conclusion. Wermelinger takes as a premise the fact that a single formalism will be simply unable to ever satisfy all requirements in every situation, and he presents three approaches making use of three different formalisms. Each approach has its own assumptions about the system and has its advantages and disadvantages.

As self-criticism, we need to mention some problems with this paper that deserve further work: first, it has been asked how we can know that the list of requirements is complete. In fact, the completeness of a list of requirements can never be determined for a generic system, but only for a specific system. Second, the survey of formalisms is incomplete. This was unavoidable due to space restrictions.

APPENDIX: SUMMARY OF REQUIREMENTS

Model Structuring Requirements

Unit encapsulation, unit compositionality, unit naming

System Modelling Requirements

- *Reconfiguration of Components*: component creation, component deletion, component replacement, component migration
- *Reconfiguration of Links*: link creation, link deletion
- *Application Behaviour*: input, output, calculation, data model, state update, control structures (sequence, parallel, alternative, iterations, recursion)
- *Interference with Runtime Support*: functional interaction, temporal interaction
- *Real-Time Information*: time model, memory model, communication model, concurrency model, interrupt model
- *Real-Time Restrictions*: scheduling restrictions, synchronization restrictions, memory restrictions
- *Fault Tolerant Behaviour*: fault models, error recovery

Analysis Requirements

Consistency, decidability, termination, deadlock freedom, scheduling feasibility, type checking, reliability analysis

REFERENCES

[1] J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996.

[2] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* To be published in 2010.

[3] J.-R. Abrial, S. Schuman, and B. Meyer. *A Specification Language.* Cambridge University Press, New York, NY, USA, 1980.

[4] N. A. Anisimov and M. Koutny. On compositionality and petri nets in protocol engineering. In *Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification*, 1996.

[5] N. C. Audsley, A. Burns, R. I. Davis, D. J. Scholefield, and A. J. Wellings. Integrating optional software components into hard real-time systems. *Software Engineering Journal*, 11(3):133–140, 1996.

[6] G. Berry and G. Boudol. The chemical abstract machine. In *Selected papers of the Second Workshop on Concurrency and compositionality*, pages 217–248. Elsevier Science Publishers Ltd., 1992.

[7] A. Bhattacharyya and J. S. Fitzgerald. Development of a formalism for modelling and analysis of dynamic reconfiguration of dependable real-time systems: A technical diary. In *SERENE 2008*.

[8] D. Bjorner and C. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer, 1978.

[9] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[10] L. Cardelli and A. D. Gordon. Mobile ambients. *Formal methods for distributed processing: a survey of object-oriented approaches*, pages 198–229, 2001.

[11] N. Dragoni and M. Mazzara. A formal semantics for the ws-bpel recovery framework - the pi-calculus way. In *WS-FM'09, Springer Verlag*, 2009.

[12] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition.* Addison-Wesley Professional, 2003.

[13] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[14] N. Kobayashi. Typical: Type-based static analyzer for the pi-calculus. http://www.kb.ecei.tohoku.ac.jp/koba/typical/.

[15] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[16] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990.

[17] M. Lemmin, K. X. He, and S. Schatz. Dynamic reconfiguration of software objects using petri nets and network unfolding. In *SMC'2000, Nashville, TN*.

[18] M. Llorens and J. Oliver. Structural and dynamic changes in concurrent systems: Reconfigurable petri nets. *IEEE Trans. Comput.*, 53(9):1147–1158, 2004.

[19] R. Lucchi and M. Mazzara. A pi-calculus based semantics for ws-bpel. *Journal of Logic and Algebraic Programming*, 70(1):96–118, 2007.

[20] P. R. Manson. Petri net theory: a survey. Technical Report 139, Computer Laboratory, University of Cambridge, June 1988.

[21] M. Mazzara. *Towards Abstractions for Web Services Composition.* PhD thesis, 2006.

[22] M. Mazzara. Deriving specifications of dependable systems: toward a method. In *Proceedings of the 12th European Workshop on Dependable Computing (EWDC 2009)*, 2009.

[23] M. Mazzara and I. Lanese. Towards a unifying theory for web services composition. In *WS-FM*, pages 257–272, 2006.

[24] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., 1982.

[25] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge University Press, 1999.

[26] J. Montgomery. A model for updating real-time applications. *Real-Time Systems*, 27(2):169–189, 2004.

[27] J. Padberg. Petri net modules. *J. Integr. Des. Process Sci.*, 6(4):105–120, 2002.

[28] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, 1962.

[29] D. Sangiorgi. Termination of processes. *Mathematical. Structures in Comp. Sci.*, 16(1):1–39, 2006.

[30] S. Schneider and H. Treharne. CSP theorems for communicating b machines. *Formal Asp. Comput.*, 17(4):390–422, 2005.

[31] S. T. H. Schneider and B. Vajar. Introducing mobility into CSP‖B. In *AVOCS 2007*.

[32] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1992.

[33] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936.

[34] W. van der Aalst. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype, 2004.

[35] M. Wermelinger. *Specification of Software Architecture Reconfiguration.* PhD thesis, Universidade Nova de Lisboa, 1999.

[36] D. Zöbel. The deadlock problem: a classifying bibliography. *SIGOPS Oper. Syst. Rev.*, 17(4):6–15, 1983.