Aspects of Formal and Graphical Design of a Bus System

Tiberiu Seceleanu
University of Turku, Dpt. of Information Technology
Turku, Finland
tiberiu.seceleanu@utu.fi

Tomi Westerlund
Turku Centre for Computer Science
Turku, Finland
tomi.westerlund@utu.fi

Abstract

This study shows the derivation of a local segmented bus arbiter from an original single segment bus arbiter. The operations are performed in the formal framework of action systems and illustrated in a graphical manner using the corresponding action systems – UML profile notations. The derivation is useful both to demonstrate the capability of preserving correctness when considering an important hardware design decision and also to identify means through which this kind of decisions can be performed in a graphical environment.

1 Introduction

The complexity of modern day digital systems, stimulated by the latest advances in technology, leads often to problems concerning the correctness of the development flow. On one hand, modular design is one of the solutions towards partially reducing the task of the designer of complex systems, while on the other hand, the employment of formal methods in system design tries to solve the aspects related to correctness. However, the pressure imposed by time-to-market aspects usually does not leave enough time for thorough analysis of designs, before they are shipped. Additionally, the often heavy mathematical apparatus behind formal method frameworks still forbids the expected wide usage of such approaches in system design.

In this situation, the Unified Modeling Language (UML) [11] emerged lately as one possible candidate design environment which provides a fast learning curve combined with certain capabilities for formal analysis. UML provides a set of "intuitive" graphical and textual description techniques that are supposed to be easily understandable for both system developers and expert users working in the application domain. Most attractive for us is the graphical notation and the possibility to adapt it to an already developed design style. The latter feature comes in extremely handy, because UML is, in fact, grossly imprecise [10].

Action Systems [1] is a state-based formalism, relying on an extended version of Dijkstra's language of *guarded commands* [4]. This study is a continuation of previous work presented in [9], which defines a UML profile for Action Systems. We use as a case study the segmented bus arbiter, also described previously in [5]. In the following sections, we show how an existent (single segment) bus arbiter description can be correctly transformed into a local segment bus arbiter. For this, we apply both the UML profile and action systems techniques.

2 The Segmented Bus Platform

The growing diversity of system-on-a-chip (SOC) devices brings up an immense number of possible interfaces. In many situations, both the system design and performance are limited by the complexity of the interconnection between the different modules and blocks that are integrated into those chips. Furthermore, different data transfer speeds are required as well as parallel transmission. A usual bus, where only one module can transmit at a time, is slow, due to large capacitive load caused by the interfaces of the modules that are attached to it and the long physical length.

A solution to the above mentioned problems is a segmented bus design combined with a globally asynchronous locally synchronous (GALS) [3] system architecture. In this approach, each distinct module of a SOC system is synchronized to a local clock, whereas interactions between those modules are arranged asynchronously. A segmented bus is a bus which is partitioned into two or more segments. Each segment acts as a normal bus between modules that are connected to it and operate in parallel with other segments. Segments can be connected / disconnected dynamically to each other in order to establish connection between modules located in different segments. Therefore, this structure is more flexible than the usual one (AMBA-like, for instance), where bridges may be interpreted as segments. All dynamically connected segments act as one single bus. Due to the segmentation of this resource, parallel transactions can take place, thus increasing the performance. Details about the

1530-1591/04 $20.00 (c) 2004 IEEE

segmented bus principles and implementation can be found in [5].

Segmented Bus Architecture. The segmented bus structure is simply illustrated in Figure 1. Every *segment* is composed of a group of masters, a group of slaves, a segment arbiter (**SA**), the physical lines (address, data, request, acknowledge and read / write lines) and an *inter-segment bridge* controller. The segments with their components act as a stand-alone busses operating in parallel, masters mostly asking services from the group of slaves placed within the confines of one segment. Occasionally, one master may require services from a slave connected to another bus segment. In this situation, the local arbitration unit forwards the request to a central arbitration module, in order to establish an inter-segment connection.

The central arbiter (**CA**) stores the information regarding the current situation of the segments: what segments are participating in an inter-segment transaction, what new requests are pending for an inter-segment access. Based on this information, the **CA** decides if there is a change in the ownership of certain segments and delivers the appropriate control signals to the involved modules.

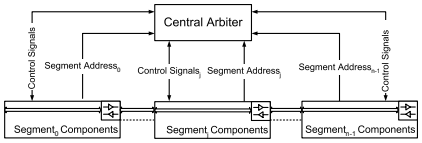


Figure 1. Segmented bus structure.

Operations on a Segmented Bus. There are three modes in which operation on a specific segment may proceed, from the point of view of local arbitration. These modes depend on the localization of the master requesting the bus and the slave. Thus, we have (i) a *local master – local slave* situation, (ii) a *local master – external slave* situation and (iii) a *external master – local or external slave*.

In all the situations, the master that is granted the access to the bus connects to the slave following a four-phase signaling protocol. The *request* part is also visible to the **SA** residing in the same segment as the master. Thus, the **SA** supervises the access of the master to the bus by counting the number of transfers, in cases (i) and (ii) above.

A more detailed block description of segment components and signals is given in figure 2.

3 Action Systems

Back and Kurki-Suonio [1] introduced the action systems formalism, providing a framework for specifying and refining concurrent programs. An *action system* (AS, henceforward) is in general a collection of *actions* or guarded commands, which are executed one at a time. The

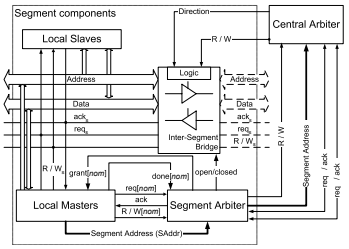


Figure 2. The Segment Control Elements.

Action Systems is used for specification and correctness preserving development of reactive systems.

An *action* A is defined (for example) by

$$\begin{array}{lll} A ::= & x := x'.R & \text{(nondeterministic assignment)} \\ & A_1 \ [] \ A_2 & \text{(nondeterministic choice)} \\ & A_1;\ A_2 & \text{(sequential composition)} \\ & A_1 \ /\!/ \ A_2 & \text{(prioritized composition)} \end{array}$$

where R is a predicate, x is a variable or a list of variables, and A_1 and A_2 are actions. Semantically, an action A is defined by the *weakest precondition* for A to establish some post-condition Q, denoted $wp(A, Q)$. In this paper, we regard an action A as being of the form $g \rightarrow S$, where g is the *guard* of the action, given by $gA \mathrel{\widehat{=}} \neg wp(A, false)$, and S is the *action body*. An action is said to be *enabled*, if its guard is $true$, *disabled* otherwise. Actions are considered *atomic*, meaning that whenever one is selected for execution, it will be completed without interference. Atomic actions may be part of *non-atomic* action compositions, in which case, the execution of the whole composition is interrupted after the execution of one of its enabled components. Additionally, the *quantified composition* of actions is defined by $[*i = 0 \ldots n : A_i] \mathrel{\widehat{=}} A_0 * A_1 * \ldots A_n$, where $*$ can be any of the allowed operators.

An *action system* is a composition of (non-atomic) actions, describing a certain behavior. As long as the system contains enabled actions, one of them may be selected for execution, after which actions in other systems may be executed.

Refinement. Action systems are meant to be designed in a stepwise manner within the *refinement calculus* framework [2]. The *refinement calculus* preserves the correctness of the actions during refinement procedure.

The (atomic) action A is said to be *(correctly) refined* by action C, denoted $A \leq C$, if $\forall q.(wp(A, q) \Rightarrow wp(C, q))$ holds. This is equivalent to the condition $\forall p, q.((p\ A\ q) \Rightarrow (p\ C\ q))$, which means that the *concrete* action C preserves every total correctness property of the *abstract* action A.

In the next sections we will make use of some rules based on refinement relations, which we introduce in the following paragraphs.

Rule 1 – Data Refinement. Assume two actions A and C with variables a, u and c, u, respectively. Let $R(a, c)$ be a

boolean relation between the variables a and c. The abstract action A is *data-refined* by the concrete action C using the *abstraction relation* $R(a, c)$, denoted $A \leq_R C$, if

$$\forall q.(R \wedge wp(A, q) \Rightarrow wp(C, \exists a.R \wedge q)) \quad (1)$$

holds. The predicate $\exists a.R \wedge q$ is a boolean condition on the program variables a and c.

This rule formalizes a very common practice in HW designs. For instance, it is usual to change data types during design steps that change the view of the system from abstract to more concrete levels. Hence, *integers* could be represented as *bit-vectors*, or simple data structures can be transformed into more complex ones, as we illustrate in forthcoming sections.

Rule 2 – Prioritized composition. The prioritized composition defined in [7] offers us the possibility to concisely impose precedence of some actions / action systems over others. It is clearly a natural solution for situations encountered in a bus based design, where at least in the arbitration process granting the next owner of the bus follows certain priority schemes.

Briefly, the prioritized composition of two actions is expressed in terms of the non-deterministic choice as: $A \;/\!/\; B \mathrel{\widehat{=}} A \;[\!]\; \neg gA \rightarrow B$. The result of interest to us is the fact that always, a nonprioritized composition $A \| B$ can be turned into a prioritized one, for instance $A \;/\!/\; B$, by merely strengthening the guard of the less important action. Hence, always:

$$A \;[\!]\; B \leq A \;/\!/\; B \quad (2)$$

Rule 3 – Distributivity of sequence over the choice. Considering the statements present in our design as conjunctive, monotonic predicate transformers [2], we benefit in our reasoning of the following rule, describing the distributivity of the sequence over the choice operator:

$$(A \;[\!]\; B); C = (A; C) \;[\!]\; (B; C) \quad (3)$$

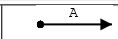
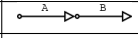
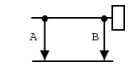
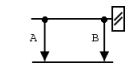
There are several other rules that we will consider in the forthcoming sections, such as the introduction of a local variable, the introduction or the removal of an empty statement, for which we do not offer detailed exemplifications. They may be found elsewhere [2, 8].

4 UML Profile for Action Systems

The UML profile for Action Systems is a graphical notation intended to ease the designer's burden by offering a visual representation of the system under development. With this graphical representation the designer can compose the system and manage large complex systems easier. The operators connecting the actions and the action systems are clearly visible in the graphical representation and thus the overall functionality becomes apparent. The profile customizes and extends specific UML type for every action system operator and defines a notation for them.

A Graphical Notation. Table 1 shows part of the graphical notation specified in the profile.

Table 1. Notation of actions

Name	Notation	AS Meaning
Atomic Action		A
Atomic Sequence		$A; B$
Non-Atomic Sequence		A**;** B
Non-Deterministic Choice		$A \;[\!]\; B$
Prioritised Composition		$A \;/\!/\; B$

The notation presented in table 1 is consistent with the action characteristics described in section 3: the full dots and arrow-heads correspond to atomic compositions, while the other line terminations correspond to non-atomic constructs. Notice that the graphical representations for the choice and prioritized composition correspond to non-atomic situations.

Within a system, the next action to be selected for execution is given by the operator semantics.

Graphical refinement. One of the intentions behind selecting UML as a partner language when developing action systems design was the existence of a graphical environment. However, images by themselves can not ensure the correctness of necessary transformations from high levels of abstraction to implementation and therefore, a textual notation of the system should be available at any time. Changes that are made in the graphical notation must be checked under refinement calculus rules, so that the target system will be a correct implementation of the original one. On the other hand, during the system design flow, there is a relatively reduced set of transformations that help the designer reason about operations to be performed in order to select among different levels of representation. Hence, in the following, we intend to map some cases of refinement rules into equivalent graphical transformation rules.

Rule G1. Choice to prioritized composition. This rule illustrates the change of a nondeterministic choice composition into a prioritized composition, as shown in figure 3.

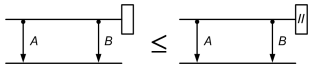


Figure 3. Prioritizing actions.

Rule G2. Distribution of sequence over choice. The relation 3 is visualized as shown in figure 4.

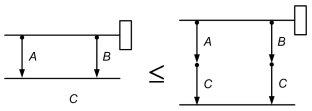


Figure 4. Atomic sequence distribution.

Rule G3. Non-atomic sequence and choice. We give as an example here a rule that is easily suggested by the graphical notation, and which can be proven within the action system framework. Hence, we can extend the above illustrated *Rule 3* to non-atomic actions, in the way presented in figure 5.

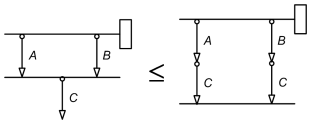


Figure 5. Non-atomic sequence distribution.

5 Derivations

We intend to start our design starting from a single segment bus, where we have the descriptions of masters, the slaves and the bus arbiter. The transformation towards a segmented bus is transparent except for the arbiter, which becomes a *segment arbiter*. The **SA** has to take into consideration now a higher priority master, represented by the **CA**. It will also have to consider idling for the period of time when the segment is just a transmission line between a winning master and its selected slave, both situated outside the boundaries of the specific **SA**segment.

In this section we concentrate on the granting activity only. The notations T and F stand for the boolean values of $true$ and $false$, respectively.

The segment arbiter. The operation of the arbiter on a single bus system consists of two jobs. One is to grant the requesting masters the access to the bus, whenever the previous owner finished the transfers. This is signaled by raising the line gr ($gr := T$). The second one is to supervise the current owner so that the number of transfers does not go over a specified limit. When this limit is reached, the master is informed that it has lost the control. Hence, the variable gr is reset ($gr := F$). With the help of the boolean variable ack the arbiter also informs the masters that the decision on the next owner of the bus is taken. If requesting masters did not yet receive access to the bus, they are supposed to keep on requesting.

While a granted master operates transfers on the bus, the arbiter must not grant any other requesting master. Hence, the supervision activity must have a higher priority than granting. Thus, we have the graphical description offered by Figure 6. This is also the starting point in transforming the single bus arbiter into a **SA**.

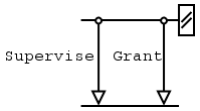


Figure 6. Single bus arbiter.

The $Grant$ Action. We start by considering that there is a single master in the system. Replicating the action in a proper manner will eventually describe the granting activity for a larger number of masters. Hence, our derivation begins with the $Grant$ action (Fig. 7). Thus, whenever a master requests the bus (req) and no other master is granted ($gr = F$), then the arbiter may give access to resources ($gr := T$) or not ($skip$ – which means that leaves $gr = F$). After this, the ack line is set to *true*. It is set back to *false* whenever the master also reset the req signal. Following this, a new granting cycle may start.

$$\begin{array}{lcl} Grant & \widehat{=} & (\ req \wedge gr = F \rightarrow (gr := T \ [\!] \ skip); ack := T) \\ & ; & \neg req \wedge ack \rightarrow ack := F \end{array}$$

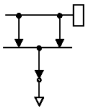


Figure 7. The initial $Grant$ action.

We apply Rule G2 to the *Grant* action and we obtain the refined version, $Grant_1$, as illustrated in Fig. 8

$$\begin{array}{lcl} Grant_1 & \widehat{=} & (\ req \wedge gr = F \rightarrow (gr := T; ack := T \ [\!] \ skip; ack := T)) \\ & & ;\neg req \wedge ack \rightarrow ack := F \end{array}$$

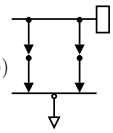


Figure 8. The action $Grant_1$.

Next, consider the relation

$$\begin{array}{lcl} R(gr, grant) & \widehat{=} & (gr = F \Leftrightarrow (grant = F \vee grant = Hold)) \\ & & \wedge\ (gr = T \Leftrightarrow grant = T) \end{array}$$

The relation R specifies how we can replace the original two-valued variable gr, with the new, three-valued, variable $grant$. This is required because, when a local master requests an external segment access, the **SA** cannot grant it without asking the **CA**. Hence, it will first place the corresponding grant line to a new value, *Hold* and will forward the request to the **CA**. When the **CA** grants the access, it informs the corresponding **SA**, which now forwards the grant to the specific master. Using R, we data refine the granting action ($Grant_1 \leq_R Grant^1$) as shown in Fig. 9.

We continue by introducing the variables which implement the communication with the **CA**, namely req_C and

$$Grant^1 \;\widehat{=}\; \big(\; req \wedge (grant = F \vee grant = Hold) \rightarrow grant := T; ack := T$$
$$[\!]\; req \wedge grant = F \rightarrow grant := Hold; ack := T \;\big)$$
$$;\neg req \wedge ack \rightarrow ack := F$$

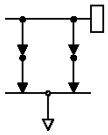


Figure 9. The action $Grant^1$.

ack_C. The first one is updated by the **SA**, while the second is only read by the **SA** and written by the **CA**. We obtain the action $Grant^2$ (Fig. 10).

$$Grant^2 \;\widehat{=}\; \big(\; req \wedge (grant = F \vee (grant = Hold \wedge ack_C)) \rightarrow$$
$$grant := T; ack := T$$
$$[\!]\; req \wedge grant = F \rightarrow grant := Hold; ack := T; req_C := T \;\big)$$
$$;\neg req \wedge ack \rightarrow ack := F$$

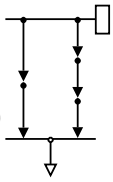


Figure 10. The action $Grant^2$.

The arbiter differentiates a local segment request from an external one by reading the slave address lines ($SAddr$) provided by the requesting master. The own address ($lsegnr$) is coded inside the **SA**. For simplicity, we denote $local \;\widehat{=}\; SAddr = lsegnr$. A new version of the *Grant* action is then obtained – Fig. 11.

$$Grant^3 \;\widehat{=}\; \big(\; req \wedge ((local \wedge grant = F) \vee (grant = Hold \wedge ack_C))$$
$$\rightarrow grant := T; ack := T$$
$$[\!]\; req \wedge \neg local \wedge grant = F$$
$$\rightarrow grant := Hold; ack := T; req_C := T \;\big)$$
$$;\neg req \wedge ack \rightarrow ack := F$$

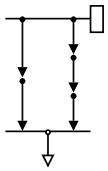


Figure 11. The action $Grant^3$.

A master that needs to transfer data to / from a slave placed in an external segment will wait considerably more than a master requesting a local resource. In order to balance this aspect, we decide to give higher priorities to such requests. In the same step, we apply *Rule* **3**. The result is represented by action $Grant^4$.

Here is the point where we take into consideration the existence of several masters within the segment. They are numbered from 0 to nom. Each master j communicates with the **SA** by means of a request signal, $req[j]$ and slave address lines $SAddr[j]$. The **SA** updates for each master a grant signal, $grant[j]$. Moreover, every master has access to the unique ack line used by the **SA** to signal termination of a granting session. Thus, we replicate the granting activity following a data refinement step which relates the initial variables req and $grant$ to their "vectorized" versions, $req[0 \ldots nom]$ and $grant[0 \ldots nom]$. We use the abstraction relation:

$$Grant^4 \;\widehat{=}\; \Big(\; (\; req \wedge grant = Hold \wedge ack_C \rightarrow grant := T; ack := T$$
$$[\!]\; req \wedge \neg local \wedge grant = F \rightarrow grant := Hold; req_C := T;$$
$$ack := T\;); \neg req \wedge ack \rightarrow ack := F \;\Big)$$
$$/\!/\; \Big(\; req \wedge local \wedge grant = F \rightarrow grant := T; ack := T$$
$$;\neg req \wedge ack \rightarrow ack := F \;\Big)$$

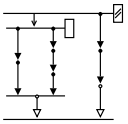


Figure 12. The action $Grant^4$.

$$R_1 \;\widehat{=}\; (req = T \Leftrightarrow \exists j \in \{0, \ldots, nom\}.req[j] = T)$$
$$\wedge\, (req = F \Leftrightarrow \forall j \in \{0, \ldots, nom\}.req[j] = F)$$

and we have $Grant^4 \;\leq_{R_1}\; Grant^5$ – Fig. 13.

$$Grant^5 \;\widehat{=}\; \Big(\; \big(\; \big[\!]\; j := 0 \ldots \mathit{nom} : req[j] \wedge grant[j] = Hold \wedge ack_C \rightarrow$$
$$grant[j] := T; ack := T \big]$$
$$[\!]\; \big[\!]\; j := 0 \ldots \mathit{nom} : req[j] \wedge \neg local \wedge grant[j] = F \rightarrow$$
$$grant[j] := Hold; ack := T; req_C := T \big] \;\big)$$
$$;\quad \neg req[j] \wedge ack \rightarrow ack := F \;\Big)$$
$$/\!/\quad \Big(\; \big[\!]\; j := 0 \ldots \mathit{nom} : req[j] \wedge local \wedge grant[j] = F \rightarrow$$
$$grant[j] := T; ack := T$$
$$;\quad \neg req[j] \wedge ack \rightarrow ack := F \big] \;\Big)$$

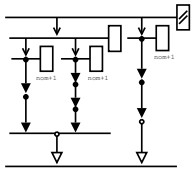


Figure 13. The action $Grant^5$.

Observe that there is one situation which is not yet exactly reflected in the above description. It corresponds to the request coming from the **CA** to the **SA**, asking access rights for another segment master, either to pass through the segment, or to access a local resource. The channel devoted to this communication is similar with the ones considered by $Grant^5$: the **CA** is just another local master from the point of view of the **SA**. However, we just want to separately identify this specific master, as we intend to place it higher in the priority list. We identify it as the "master[0]". We assign to this master the highest priority. Notice further that, actually, the **CA** does not provide requested segment addresses. Hence, we may assume that it always requests *local* access, therefore, the corresponding *grant* line can not be placed on *Hold*. At the same time, we decide to replace the variables $req[0]$ and $grant[0]$ by req_O and ack_O, respectively (Fig. 14).

$$Grant^7 \;\widehat{=}\; \Big(\; \big[\!]\; j := 1 \ldots \mathit{nom} : req[j] \wedge grant[j] = Hold \wedge ack_C$$
$$\rightarrow grant[j] := T; ack := T$$
$$[\!]\quad req[j] \wedge \neg local \wedge grant[j] = F$$
$$\rightarrow grant[j] := Hold; ack := T; req_C := T$$
$$;\neg req[j] \wedge ack \rightarrow ack := F \big] \;\Big)$$
$$/\!/\; \Big(\; req_O \wedge ack_O = F \rightarrow ack_O := T; ack := T$$
$$;\neg req_O \wedge ack \rightarrow ack := F \;\Big)$$
$$/\!/\; \big[\!]\; j := 1 \ldots \mathit{nom} : req[j] \wedge local \wedge grant[j] = F$$
$$\rightarrow grant[j] := T; ack := T$$
$$;\neg req[j] \wedge ack \rightarrow ack := F \big]$$

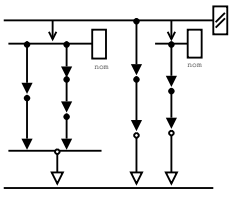


Figure 14. The action $Grant^7$.

The segment arbiter. An analysis similar to the one presented above (for the $Grant$ action) is performed for the $Supervise$ action. The full representation of the **SA** is illustrated in figure 15, where we also named the subcomponents of the $Grant^7$ action. The action system description is given in [6].

$$
\begin{aligned}
LAckF &\;\widehat{=}\; \neg req[j] \wedge ack \rightarrow ack := F \\
LAckF_O &\;\widehat{=}\; \neg req_O \wedge ack \rightarrow ack := F \\
RoAckC &\;\widehat{=}\; req[j] \wedge grant[j] = Hold \wedge ack_C \rightarrow grant[j] := T; \\
&\qquad ack := T \\
RoGr &\;\widehat{=}\; req[j] \wedge \neg local \wedge grant[j] = F \rightarrow grant[j] := Hold; \\
&\qquad ack := T; req_C := T \\
EGr &\;\widehat{=}\; req_O \wedge ack_O = F \rightarrow ack_O := T; ack := T \\
LGr &\;\widehat{=}\; req[j] \wedge local \wedge grant[j] = F \rightarrow grant[j], ack := T
\end{aligned}
$$

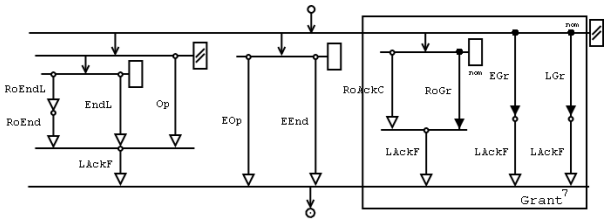


Figure 15. The segment arbiter.

6 Conclusions

In this study we have shown how the action systems formalism is applied to correctly derive a segment arbiter specification from a single segment arbiter. The work is part of a larger project that analyzes the realization of a segmented bus, starting from high levels of abstraction down to implementation.

On the way, we also applied the graphical notations of the action systems profile for UML. We described how the precise rules of refinement can be translated into the graphical environment provided by this profile. However, we may not say that one can directly use the visual environment to fulfill all the requirements of a given design project. The action systems representation should always be aside and consulted at every step. A small step towards the relaxation of this situation is illustrated by the translation of several refinement steps into their graphical equivalents. Nevertheless, for the possible necessary data refinement steps we found yet no solution as to their UML representation.

In this direction, further work concentrates on analyzing possibilities to adapt the techniques of the action systems framework to the environment provided by the Object Constraint Language [11]. Provided the changes in system representation are correctly performed, the graphical description transformations, as the ones presented in the previous section (Fig. 7 to Fig. 14) can be easily automated by using OCL specifications. Thus, one could think of building a design methodology that would take advantage of the features offered by the AS–UML profile.

Acknowledgements. The work presented in this study was partly funded by the ITEA – *Prompt to Implementation* project.

References

[1] R. J. R. Back and R. Kurki-Suonio. Distributed Cooperation with Action Systems. *In ACM Transactions on Programming Languages and Systems*, Vol. 10, No. 4.1988, pp. 513-554.

[2] R. J. R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction.* Springer–Verlag, 1998.

[3] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems.* PhD thesis, Standford University, 1984.

[4] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall International, 1976.

[5] T. Seceleanu, J. Plosila, P. Liljeberg. On-Chip Segmented Bus: A Self Timed Approach. Proceedings of the 15^th IEEE ASIC/SOC Conference, 2002, pages 216-221.

[6] T. Seceleanu, T.Westerlund. Segment Arbiter As Action System. Proceedings of SCS International Symposium, 2003, pp 249-252.

[7] E. Sekerinski, K. Sere. *A Theory of Prioritizing Composition.* The Computer Journal, VOL. 39, No 8, pp. 701-712. The British Computer Society. Oxford University Press.

[8] K. Sere. *Stepwise Derivation of Parallel Algorithms.* Ph.D. Thesis, Abo Akademi, Turku, Finland, 1990.

[9] T.Westerlund, T. Seceleanu. UML Profile for Action Systems. To appear as a TUCS technical report, 2003.

[10] J. Whittle. Formal Approaches to Systems Analysis Using UML: An Overview. In *Advanced Topics in Database Research, ed. Keng Siau, pps. 324-341.*

[11] OMG Unified Modeling Language Specification, ver. 1.4, September, 2001.